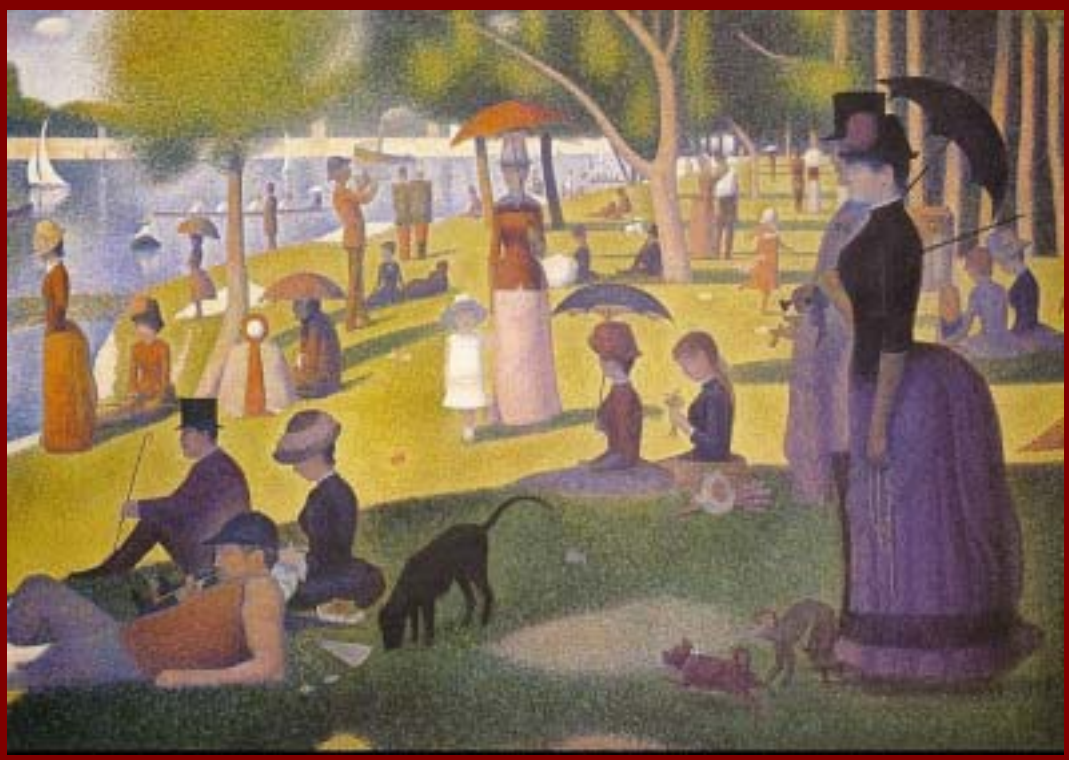
AIX DIY (Special Edition)

IBM

Migrating Solaris Applications to AIX Quick Reference

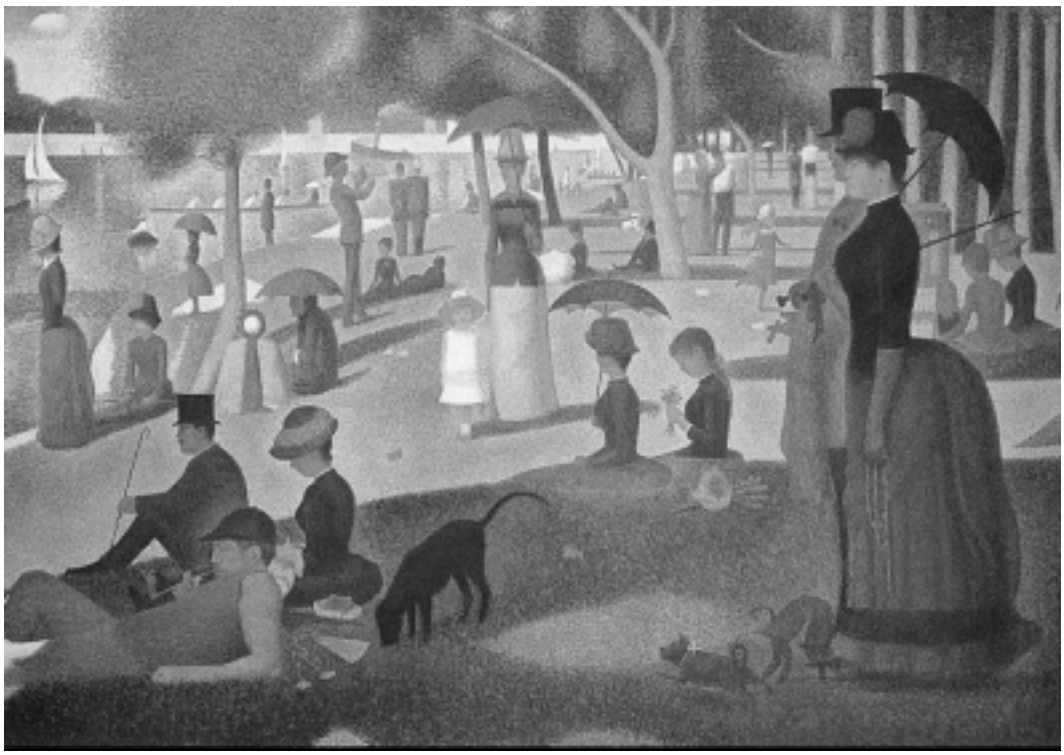


2003년 6월

한국아이비엠주식회사
기술영업지원센타

AIX DIY (Special Edition)

Migrating Solaris Applications to AIX Quick Reference



2003년 6월

한국아이비엠주식회사
기술영업지원센타

서 문

pSeries의 성장과 더불어 최근 타 벤더의 UNIX 플랫폼으로부터 AIX로 이전하는 고객들로부터 포팅에 관한 많은 질문 및 지원요청이 생겨나고 있습니다. 소프트웨어 업체에서 제공하는 제품이 아니라 고객사에서 개발된 특정 소프트웨어를 사용해야 할 때는, 새로운 시스템에서 고객의 소프트웨어가 제대로 작동될 수 있도록 하는 포팅은 필수적인 과정입니다.

IBM은 최근 AIX5L 5.2를 새롭게 발표하였습니다. AIX5L은 IBM POWER아키텍처뿐만 아니라 인텔 아이테니엄 시스템까지 지원할 수 있는 64비트 오픈 유닉스 시스템입니다. AIX 5.2에서는 기존의 Linux Affinity와 Linux toolbox for AIX CD를 제공하는 것 외에 Linux Affinity를 좀 더 강화하여 개발자들에게 향상된 편이성을 제공할 수 있게 되었습니다.

AIX사용자 그룹 및 기술지원 전문가들을 효율적으로 지원하기 위해서 한국 IBM 기술영업지원센터에서는 1999년부터 AIX DIY라는 책을 만들어 왔습니다. 여기에 덧붙여 특정분야의 내용을 깊게 다루기 위해 AIX DIY Special Edition이 나오고 있으며 현재 Porting Guide II - Porting Tip AIX DIY Special Edition까지 출간되어 있습니다.

고객의 요구사항과 AIX5L의 등장으로 인하여 포팅의 중요성이 점점 더 강조되는 상황에서 이 가이드를 발간하게 되었습니다. 이 가이드는 2001년도 출간된 Solaris to AIX porting guide를 보충하기 위해 제작되었으며 기존 Solaris 사용자 및 개발자를 위해 포팅예제와 명령어 레퍼런스 위주로 꾸며졌습니다. AIX상으로 포팅하는 데 어려움을 겪으시는 고객과 Business Partner사 여러분께 도움이 되기를 기대합니다.

한국 IBM 기술 영업 지원 센터　이 재 웅

특기사항

이 책에 기재되어 있는 정보의 사용 또는 테크닉의 실행은 고객의 평가와 사용하고 있는 작동환경의 통합 능력에 따라 고객의 책임 하에 수행되고 있습니다. 기재된 정보 및 테크닉은 둘 다 IBM에 의해 특정 상황 하에서의 정확성은 확인되어 있지만, 어떤 환경에서 항상 그것과 동일한 결과를 얻을 수 있다고는 할 수 없습니다. 고객 자신의 환경에서 이들의 테크닉을 사용할 경우에는 고객의 책임 하에 수행되고 있습니다. 의문사항은 김세희(kseh@kr.ibm.com)씨에게 문의하여 주시기 바랍니다.

차 례

시작하며 7
간단한 용어 정의 7
개론 8
AIX 소개 8
Solaris와 AIX의 차이점 9
숫자값에 대한 정의 9
LIBPATH 환경변수 9
데이터 타입 9
컴파일러 옵션 11
코드 생성관련 옵션 12
디버깅 옵션 12
부동 소수점 관련 옵션 14
언어 옵션 14
라이브러리 옵션 14
성능 옵션 16
기타 옵션 16
라이브러리와 아카이브 17
공유 라이브러리와 공유 오브젝트를 다룰 때 AIX와 Solaris의 차이점 18
링킹과 로드시 AIX와 Solaris의 차이점 18
라이브러리 검색 경로 생성과 사용방법에서의 AIX와 Solaris의 차이점 19
라이브러리 검색 경로에서의 AIX와 Solaris의 차이점 21
메모리 세그먼트 21
32bit 주소 공간 22
런타임 이슈 - 32bit 대 64bit 커널 23
AIX 4.3에서 컴파일한 32-bit, 64-bit 어플리케이션을 AIX 5.1에서 수행하기 24
성능 튜닝 24
쓰레드 프로그래밍 24
POSIX 쓰레드 라이브러리 대 Solaris 쓰레드 라이브러리 25
시그널 27

포팅 이슈 27
컴파일러 에러 27
표준 C++ 헤더파일(.h 확장자가 있는 경우와 없는 경우) 27
네임스페이스 27
콘테이너 28
캐스팅 28
fstream 28
fdopen 29
char constant 참조 29
const 참조 31
참조자로 반환된 오브젝트 저장하기 32
Copy constructors, equals operator, comparison operator 33
메쏘드 네이밍 33
friend 사용법 33
static friend 사용법 34
class qualifier 34
#endif 다음 텍스트 34
링크드 리스트 34
C++ 스타일 주석문 36
ENONET 에러 36
MAXHOSTNAMELEN 매크로 36
시스템 콜 차이 36
Unresolved seteuid(), setegid() 36
프로세스 정보 알아내기 36
시그널 37
cftime() 38
dladdr() - 심볼릭 어드레스 계산하기 38
프로그래밍 이슈 41
IPC 제한 42
공유 메모리 제한과 EXTSHM 사용법 43
RPC 44
세마포어와 RealTime 옵션 45
예제 45

세마포어 해법 46
스택 프레임 이슈 46
/proc 파일 구조 49
메모리 매핑 파일 49
자바 이슈 51
쓰레드 모델 이슈 51
성능 51
지원여부 53
결론 53
부록 Solaris 와 AIX 명령어 비교 54
패키징 54
설치와 업그레이드 55
부팅/셧다운 56
사용자 관리 57
디바이스 관리 58
네트워크 관리 59
프린터 관리 59
파일시스템 관리 60
디스크 관리 61
logical volume 관리 61
Troubleshooting 63
참고문서 65

Migrating Solaris Applications to AIX - Quick Reference 에서는 Solaris에서 AIX플랫폼으로 어플리케이션을 마이그레이션할 때 발생할 수 있는 여러 이슈들에 대해 다루고 있습니다. 컴파일러 옵션에서부터 세마포어, 스택 프레임에 이르기까지 다양한 주제에 대해 다루고 있으며 특히 C++에 대해 집중적으로 다루고 있습니다. 여기에 다양한 시나리오와 이에 따른 Solaris에서 AIX로 마이그레이션하는 다양한 예제에 대해서도 수록하고 있습니다. 그리고 부록으로 Solaris와 AIX의 OS 명령어를 비교하고 있으므로 명령어 레퍼런스로 사용할 수 있습니다. AIX에 대한 더 자세한 정보는 참고문서에서 언급한 문서를 자세히 읽어보시기 바랍니다.

시작하며

Solaris에서 수행하는 어플리케이션을 AIX에서도 수행하려면 대부분의 경우 재컴파일 과정을 거치면 된다. 그러나 예외는 항상 발생할 수도 있으므로, 이 문서에서는 이를 해결하기 위한 다양한 상황들이나 간단히 소스코드를 수정해야 할 상황에 대해 알아보도록 한다.

이 문서는 AIX 5.1 을 기준으로 하며, AIX 5.1은 현재 PowerPC와 Itanium프로세서 용으로 출시되어 있다. 여기에서는 PowerPC용 AIX에서 포팅하는 상황을 가정한다.

간단한 용어 정의

다음은 여기에서 등장하게 될 용어에 대한 간단한 설명이므로 잠깐 읽어보도록 하자.

ILP32	32비트 어플리케이션 소스. 이 모델에서 C/C++의 int, long, 포인터 크기는 모두 32비트이다.
LP64	64비트 어플리케이션 소스. 이 모델에서 C/C++의 int는 32비트이고, long, 포인터의 크기는 64비트이다.
AIX5L	PowerPC, 아이테니엄 기반 시스템에서 수행되는 AIX OS의 버전 5
Little Endian(LE)	데이터 워드에서 바이트의 배치가 오른쪽에서 시작해서 왼쪽에서 끝난다.
Big Endian(BE)	데이터 워드에서 바이트의 배치가 왼쪽에서 시작해서 오른쪽에서 끝난다. AIX와 Solaris 모두 Big Endian을 따르고 있다.

개론

대부분 어플리케이션은 한가지 이상의 하이 레벨 프로그래밍 언어로 작성한다. 매크로 어셈블리나 어셈블리만으로 만든 어플리케이션은 플랫폼이 바뀌는 경우 완전히 새로 만들어야 하므로 이 경우는 제외하고, 여기서는 C/C++ 로 만든 프로그램을 포팅하는 이슈에 대해서만 다루겠다.

대부분의 C, C++ 로 만든 프로그램의 경우 "잘 만들어진" 소스는 AIX상에서 거의 바꾸지 않고 바로 컴파일해서 사용할 수 있다. 여기에서 "잘 만들어진"의 의미는 다음과 같은 프로그래밍 기법들을 잘 지키고 있음을 의미한다.

- ANSI/ISO C/C++ 표준을 잘 따르고 있는지
- 소프트웨어 개발 단계상 디자인, 개발, 유지보수에서 포팅을 염두에 두었는지
- 코드 전체에 걸쳐 함수 프로토타입 선언을 제대로 사용했는지

한 플랫폼에서 다른 플랫폼으로 어플리케이션을 포팅하려면 재컴파일, 링킹, 디버깅, 테스트 그리고 필요에 따라 성능 튜닝의 과정을 거쳐야 한다. 표준안을 충실히 따르고 플랫폼의 특정 아키텍쳐에 의존하지 않도록 만든 프로그램의 경우는 약간만 수정하면 쉽게 포팅할 수 있다.

AIX 소개

AIX는 코어 비즈니스용 어플리케이션 및 24x365를 지원하기 위한 IBM의 UNIX® 운영체제이다. 현재 지원되는 AIX 버전은 4.3.3, 5.1, 5.2 가 있다.

AIX의 강점은 오랜 기간동안 여러 분야의 산업에서 사용되면서 증명된 안정성과, IBM의 비교적 작은 시스템(프로세서 1개 장착 시스템)부터 대용량 병렬처리 시스템인 POWERparallel® (SP)에 이르기까지 공통으로 지원할 수 있는 다양한 환경적응성에 있다. AIX 5.1에서는 이에 추가하여 다음과 같은 기능을 가진다.

- 가장 폭 넓게 사용되는 32bit 커널과 이에 64bit 커널 추가
- 32bit/64bit API 지원
- Linux 플랫폼 상에서 개발된 프로그램을 AIX 플랫폼으로 쉽게 포팅할 수 있도록 도와주는 Linux affinity
- 복잡한 작업을 위해 자원을 쉽게 할당할 수 있도록 도와주는 Workload Manager
- 간단한 시스템 관리도구 - SMIT(System Management Interface Tool)과 WSM(Web-based

System Management)

Solaris와 AIX의 차이점

다음은 AIX와 Solaris의 차이점에 대해 설명하고 있다.

숫자값에 대한 정의

숫자값에 대한 모든 정의는 다음 파일에서 찾아볼 수 있다.

AIX	`/usr/include/sys/limits.h` `/usr/include/float.h`
Solaris	`/usr/include/limits.h` `/usr/include/float.h`

시스템 아키텍쳐상의 차이로 숫자값의 최대값, 최소값 정의가 다를 수 있으므로 헤더 파일을 잘 읽어보길 바란다.

LIBPATH 환경변수

라이브러리 경로를 가리키는 환경변수는 Solaris에서는 `LD_LIBRARY_PATH` 이고 AIX는 `LIBPATH`를 사용한다.

데이터 타입

AIX (32/64 bit)에서 double 데이터 타입은 8바이트이며, 4byte 바운더리로 얼라인된다. 더 자세한 정보는 다음 테이블 1.0과 1.1을 참조하도록 하자.

데이터 타입	AIX		Solaris
	32bit 소스	64bit 소스	64bit 소스
char	8	8	8
short	16	16	16
int	32	32	32
float	32	32	32
pointer	32	64	64
long	32	64	64
long long	64	64	64
double	64	64	64

long double	64	64 128(LONGDOUBLE128을 정의해서 컴파일 했을 때)	128

테이블 1.0 : C 데이터 타입 (데이터 타입 단위는 비트(bit))

데이터 타입	AIX (64bit)	Solaris (64bit)
char	1	1
short	1/2	1/2
int	4	4
float	4	4
pointer	8	8
long	8	8
long long	8	8
double	8	8

테이블 1.1 : 데이터 얼라인먼트 (데이터 타입 크기 단위는 바이트(byte))

대부분의 RISC 시스템은 메모리 내 데이터 얼라인먼트에 매우 민감하게 영향을 받으며 특정 바운더리에 의해 성능이 바뀔 수 있다. 그러나 특별히 정해진 원칙은 없다. 예를 들어 struct형 데이터에는 특정 바운더리가 최선의 성능을 내지만 array형 데이터에는 그다지 좋지 않을 수도 있다. 이 모두 메모리 공간이 효율적으로 사용되지 않거나 캐시에 데이터가 올라오지 않아서 발생할 수 있는 문제들이다.

AIX 컴파일러에서는 얼라인먼트 관련하여 4가지의 컴파일러 옵션을 제공한다.

-qalign	의미
Power (or full)	PowerPC 얼라인먼트 규칙을 따른다.
Twobyte	Apple 매킨토시 얼라인먼트 규칙을 따른다.
Packed	packed 얼라인먼트 규칙을 따른다.
Natural	struct 데이터의 멤버는 모두 자신의 원래 데이터 바운더리에 따라 매핑된다.

테이블 1.2 : 컴파일러에서 지원되는 얼라인먼트 관련 옵션

디폴트 얼라인먼트는 power이다. 특정 옵션을 지정하지 않았을 때 자연스럽게 얼라인되는 규칙도 power를 따르고 있는데, 이 경우 struct나 union의 첫번째 멤버가 아닌 double 이나 long double 은 제외한다. 따라서 qalign 옵션을 따로 natural로 정의하지 않은 경우 long, double, long double은 워드 얼라인을 따르게 된다. double은 더블워드(8바이트)얼라인되고, long double은 롱 더블워드 (16바이트) 얼라인을 따른다.

컴파일러 옵션

AIX는 C/C++ ANSI 표준안을 지원하고 있다. AIX용 C/C++ 최신 컴파일러는 VisualAge® for C and C++ V6.0(VAC 6.0)이며 C/C++의 최신 정의를 모두 지원한다.

컴파일러 바이너리 C는 `/usr/bin/vac` 에 , C++은 `/usr/bin/vacpp` 에 설치하고, 문서파일의 디폴트 위치는 C는 `/usr/vac/html/en-US/doc/index.htm`, C++은 `/usr/vacpp/bin/vacpphelp` 이다. 컴파일러는 `/etc/vac.cfg` 파일에 여러 가지 디폴트 환경을 설정해 놓았는데, 이 디폴트 환경은 어떤 드라이버 프로그램을 호출하는가에 따라 다르게 사용된다. 예를 들어 UNIX98 쓰레드 라이브러리를 사용하는 ANSI C 프로그램을 컴파일하려면 `/usr/vac/bin/xlc_r` 를 사용해야 하고, POSIX Draft 7 쓰레드 라이브러리를 사용하는 ANSI C 프로그램을 컴파일하려면 `/usr/vac/bin/xlc_r7`를 사용해야 한다. 그리고 쓰레드를 사용한 C++ 프로그램을 컴파일하려면 `/usr/vacpp/bin/xlC_r` 를 사용해야 한다.

AIX와 Solaris 모두 디폴트로 컴파일 후 32bit 오브젝트 코드를 생성한다. Solaris에서는 `-xarch=v9` 옵션을 사용해서 컴파일하면 64bit 환경을 따르는 64bit 코드를 생성하고, AIX에서는 `-q64` 옵션을 사용한다. AIX에서 64bit 코드를 생성하는 또 다른 방법으로 환경변수 `OBJECT_MODE`를 64로 설정한 다음 (`OBJECT_MODE=64`) 컴파일하는 방법이 있다. 이때 `-qwarn64` 옵션을 사용하면 64bit 관련 마이그레이션 문제가 발생할 염려가 있을 경우 경고해준다.

64bit 어플리케이션과 링크시키는 라이브러리는 반드시 64bit여야 한다.(특히 써드파티제품의 라이브러리인 경우 주의해야 한다.) AIX상에서 32bit 모듈이나 라이브러리의 오브젝트 모드를 보려면 다음 명령어를 사용해야 한다.

```
dump -X32 -ov <file_name>
```

<file_name> 에는 `libxyz.a`와 같이 어떤 오브젝트 모듈의 이름이라도 사용할 수 있다. 명령어의 결과에서 "`XCOFF`" 문자열을 찾아보도록 하자.

64bit 용 모듈이라면 다음 명령어를 사용한다.

```
dump -X64 -ov <file_name>
```

아래 테이블에서 Solaris C++ 컴파일러(Forte C++)와 AIX C++ 컴파일러(VisualAge C++ 5.0)의 옵션 중 일부분은 상호 참조해야 한다. Solaris C/C++ 컴파일러와 AIX 컴파일러에서 사용하는 옵션끼리의 관계를 좀더

자세히 알고 싶다면 IBM 레드북 AIX 5L Porting Guide 의 부록 C 를 읽어보기 바란다. 더 자세한 정보는 http://ibm.com/software/ad/vacpp/library.html 에 나와 있는 매뉴얼을 참고할 것.

주의: 매뉴얼의 내용은 AIX 5L Porting Guide 레드북보다 더 최신 내용이다.

- VisualAge C++ Professional for AIX 5.0 Batch Compiler and Other Tools
- VisualAge C++ Professional for AIX 5.0 Getting Started
- VisualAge C++ Professional for AIX 5.0 Migration Guide

시간이 있으면 다음 문서도 읽어보자.

- ld 명령어 man pages

코드 생성관련 옵션

Solaris 컴파일러 플랙	AIX 컴파일러 플랙	기능
-compat	-qlanglvl=compat366,.. (다른 호환 모드도 사용할 수 있다)	컴파일러의 호환성 모드를 결정한다.
+d	N/A	C++ 인라인 함수를 확장하지 않는다.
+e{0|1}	N/A	가상 테이블 생성을 결정한다.
-g	-g	디버거에서 오브젝트 코드를 사용할 수 있도록 컴파일한다.
-KPIC, -kpic	VisualAge 5 는 디폴트로 위치무관(position-independent) 코드를 생성한다.	위치무관(position-independent) 코드를 생성한다.
-mt	xlC_r	멀티쓰레드 코드를 컴파일하고 링크한다.
-xcode=a	N/A	코드 주소 공간을 지정한다.
-xMerge	N/A	텍스트 세그먼트와 데이터 세그먼트를 합친다.
-z arg	-b arg	링커 옵션

디버깅 옵션

Solaris 컴파일러 플랙	AIX 컴파일러 플랙	기능
-dryrun	-#	드라이버 프로그램에서 실제 컴파일러로 넘기는 옵션만을 보여주고 실제 컴파일

		은 수행하지 않는다.
-E	-E	프리프로세서만 수행하고 결과는 표준출력으로 보낸다.
-g	-g	디버거에서 오브젝트 코드를 사용할 수 있도록 컴파일한다.
-g0	N/A	디버거에서 오브젝트 코드를 사용할 수 있도록 컴파일한다. 단 인라인을 허용한다.
-H	-qshowinc	인클루드된 파일의 경로를 보여준다.
-keeptmp	N/A	컴파일 중간결과로 만들어지는 파일을 삭제하지 않고 그냥 둔다.
-migration	N/A	이전 버전의 컴파일러에서 마이그레이션 하는 정보를 어디서 얻을 수 있는지 알려준다.
-P	-P	소스 파일을 프리프로세서로만 처리한 다음 결과를 .i 파일로 만든다.
-Qoption	-WProgram, option	해당 옵션을 바로 컴파일 단계의 프로그램에게 넘긴다.
-readme	xlC	온라인 README 파일을 보여준다.
-s	-s	실행파일에서 심볼 테이블 부분을 삭제해서, 코드 디버깅을 방지한다.
-temp=dir	N/A	임시파일을 저장할 위치를 지정한다.
-verbose=vlst	-qinfo	컴파일 단계에서 알려주는 정보의 수준을 결정한다.
-xhelp=flagsk	xlC	대략의 컴파일러 옵션 목록을 보여준다.
-xildoff, -xildon	N/A	Incremental Linker 사용을 금지/허용 한다.
-xs	N/A	오브젝트 파일(.o 파일)없이 dbx로 디버깅을 할 수 있도록 허용한다.
-xsb	N/A	Sun Workshop 소스 브라우저에서 사용할 테이블 정보를 생성한다.
-xsbfast	N/A	소스 브라우저 정보를 생성한다. 컴파일은 수행하지 않는다.

부동 소수점 관련 옵션

Solaris 컴파일러 플랙	AIX 컴파일러 플랙	기능
-fns[= {no|yes}]	-qfloat=suboption	SPARC 비표준 부동 소수점 모드를 비활성화/활성화한다.
-fround=r	-qfloat=suboption	초기부터 IEEE 근사값 모드를 허용하도록 지정한다.
-fsimple=n	-qfloat=suboption	부동 소수점 최적화를 우선적으로 수행한다.
-ftrap=tlst	-qfloat=suboption	초기부터 IEEE 트랩 모드를 허용하도록 지정한다.
-xlibmieee	-qfloat=suboption	예외가 발생할 때 수학루틴에서 libm이 IEEE 754 값을 반환하도록 한다.

언어 옵션

Solaris 컴파일러 플랙	AIX 컴파일러 플랙	기능
-compat	-qlanglvl=compat366,… (다른 호환성 관련 옵션도 사용할 수 있다)	컴파일러에서 사용하는 여러 중요한 호환성 모드를 설정할 수 있다.
-features=alst	-qlanglvl=extended (그러나 C++ 의 생성자와 관련된 여러 복잡한 옵션은 사용할 수 없다)	C++과 관련된 여러가지 언어 특성을 활성화/비활성화 시킬 수 있다.
-xtrigraphs	N/A	trigraph sequences 인식을 활성화 시킨다.

라이브러리 옵션

Solaris 컴파일러 플랙	AIX 컴파일러 플랙	기능
-Bbinding	-bbinding	심볼릭 링킹, 동적 링킹, 정적 링킹을 선택할 수 있다.
-d{y|n}	N/A	전체 실행 파일에 동적 라이브러리를 허용/금지 할 것인지 지정할 수 있다.
-G	-G (-qmkshrobj 과는 약간 차이가 있다)	실행 파일 대신 동적 공유 라이브러리를 생성한다.

-hname	N/A	생성된 동적 공유 라이브러리에 이름을 할당한다.
-I	N/A	ld가 LD_LIBRARY_PATH에 세팅된 값들을 참조하지 않도록 한다.
-Ldir	-Ldir	라이브러리를 찾을 디렉토리를 추가한다.
-llib	-llib	링커가 라이브러리를 찾을 때 liblib.a나 liblib.so도 찾도록 한다.
-library=llist	N/A	컴파일, 링킹시 특정 라이브러리와 지정된 파일을 강제로 포함시킨다.
-mt	xlC_r	멀티쓰레드 코드를 컴파일하고 링크한다.
-norunpath	N/A	라이브러리 경로를 실행파일에 포함하지 않는다.
-Rplst	N/A	동적 라이브러리를 찾는 경로를 실행파일에 포함한다.
-staticlib=llst	-bstatic -llib -bdynamic	어떤 C++ 라이브러리를 정적으로 링크할 것인지 지정한다.
-xar	N/A	아카이브 라이브러리를 생성한다.
-xbuiltin[=opt]	N/A	표준 라이브러리 함수를 호출할 때 고수준의 최적화를 허용/금지한다.
-xia	N/A	적절한 수학 라이브러리와 링크시키고, 부동 소수점 환경을 설정한다.
-xlibmieee	N/A	예외가 발생할 때 수학루틴에서 libm이 IEEE 754 값을 반환하도록 한다.
-xlibmil	N/A	최적화를 위해서 선택한 몇몇 libm 라이브러리 루틴을 인라인시킨다.
-xlibmopt	N/A	최적화된 수학 루틴 라이브러리를 사용한다.
-xlic_lib=sunperf	N/A	SPARC: Sun Performance Library와 링크한다.
-xnolib	N/A	디폴트 시스템 라이브러리와 링크하지 못하도록 한다.

-xnolibmil	N/A	수학 라이브러리 루틴은 인라인하지 않는다.
-xnolibmopt	N/A	수학 루틴 라이브러리를 사용하지 않는다.
-xlang=l[,l]	N/A	적절한 실시간 라이브러리와 특정 언어에 맞는 실시간 환경설정을 포함시킨다.

성능 옵션

Solaris 컴파일러 플랙	AIX 컴파일러 플랙	기능
-xarch=isa	-qarch=pwr,…	목표하는 특정 아키텍처 명령어 세트를 지정한다.
-xcache=c	-qcache	SPARC: 최적화를 위해 캐시의 특정 특성을 지정한다.
-xchip=c	N/A	목표하는 프로세서 칩을 지정한다.
-xF	N/A	링커가 함수의 순서를 재조정하도록 한다.
-xinline=flist	-qinline / -Q	최적화를 위해 사용자가 작성한 특정 루틴을 인라인시킬 것인지 지정한다.
-xipo[={0|1}]	-qipa	프로시져간 관계를 고려한 최적화를 수행한다.
-xOlevel	-O (-O2, -O3, -O4)	최적화 수준을 지정한다.
-xsb	N/A	Sun Workshop 소스 브라우저에서 사용할 테이블 정보를 생성한다.
-xsbfast	N/A	소스 브라우저 정보를 생성한다. 컴파일은 수행하지 않는다.

기타 옵션

Solaris 컴파일러 플랙	AIX 컴파일러 플랙	기능
-c	-c	컴파일만 수행한다. 결과로 오브젝트 파일(.o 파일)만을 생성하고 링킹은 수행하지 않는다.
-H	N/A (그러나 -qsource -qshowinc 옵션	인클루드된 파일의 경로를 보여준다.

	을 사용하면 리스트 파일에 인클루드된 소스 파일을 적어준다)	
-o [파일이름]	-o [파일이름]	생성된 결과물이나 실행파일의 이름을 [파일이름]으로 만들어 준다.
-D[심볼이름]	-D[심볼이름]	프리프로세서에게 심볼이름을 정의해서 알려준다.
-U[심볼이름]	-U[심볼이름]	프리프로세서에게 심볼 이름의 정의를 해제해서 알려준다.
-xM	-M / -qmakedep	Makefile에서 사용할 파일간의 의존관계 정보를 알려준다.
-p	-p	프로파일링 도구 prof에서 필요한 정보를 모을 수 있도록 오브젝트 코드를 생성한다.
-pg	-pg	프로파일링 도구 gprof에서 필요한 정보를 모을 수 있도록 오브젝트 코드를 생성한다.
-xa	N/A	프로파일링 도구에서 사용할 수 있도록 오브젝트 코드를 생성한다.
-xprofile=tconv	N/A	실시간 프로파일링 데이터를 모으거나 이 정보를 이용하여 최적화를 수행한다.
-instances=a	N/A	템플리트 인스턴스의 위치나 링킹에 관련된 사항을 조정한다.
-ptipath	N/A	템플리트 소스를 찾을 추가 디렉토리를 지정한다.
-template=wlst	N/A	여러 템플리트 옵션을 허용/금지한다.
+w	N/A	필요에 따라 추가적인 경고 메시지를 보여준다.
-w	-w	경고 메시지를 보여주지 않는다.
-xtime	-qphsinfo	각 컴파일 단계에서 사용된 수행시간을 알려준다.

라이브러리와 아카이브

AIX에서 아카이브는 여러가지 다른 종류의 오브젝트를 포함할 수 있다. 예를 들어 `libc.a`파일은 32bit 버전

/64bit 버전 2개의 .o 파일을 포함하고, 32bit 공유 라이브러리, 64bit 공유 라이브러리도 포함되어 있다. 어플리케이션(32bit, 64bit 모두 가능하다)은 이 아카이브를 정적, 혹은 동적으로 링크할 수 있다. AIX 링커는 이러한 면에서 Solaris와 약간 다르게 동작한다.

AIX에서 심볼 위치 결정은 링킹시 결정된다. 만약 컴파일시 명령행에서 지정해준 공유 오브젝트(혹은 모듈)에 심볼이 정의되어 있다면, 링커는 심볼 이름과 모듈이름을 프로그램의 로더 섹션에 기록해둔다. 프로그램을 수행하게 되면 심볼이 결정되어야 하고, 이 심볼의 결정은 미리 링킹시 기록해둔 모듈에서 찾게 된다.

공유 라이브러리와 공유 오브젝트를 다룰 때 AIX와 Solaris의 차이점

AIX와 Solaris는 공유 오브젝트에 대해 서로 다른 시각을 가진다. AIX에서 공유 모듈은 오브젝트 파일일 수도 있고 아카이브 파일 멤버일 수도 있다. 위에서도 기술했듯이, 한 개의 공유 아카이브 라이브러리는 여러 개의 파일을 멤버로 가질 수 있고, 각각의 파일은 공유 모듈일 수도 있고 일반적인 오브젝트 파일이어도 된다.

Solaris에서는 공유 라이브러리는 항상 일반적인 오브젝트 파일이어야만 하는데, 이 오브젝트 파일은 직접 소스 파일을 특수한 옵션을 사용하여 컴파일하고 다른 방법으로 링크시킨 파일이다. 보통 AIX에서 시스템 공유 라이브러리는 아카이브 파일이며, 시스템 라이브러리가 변경되었을 때 어플리케이션에 최소한의 영향만을 주고 수정할 수 있도록 편의를 제공한다.

링킹과 로드시 AIX와 Solaris의 차이점

AIX에서는 특정한 심볼을 참조하는 경우에만 해당되는 공유 오브젝트를 링커의 입력으로 지정하여 결과물 파일 안에 저장하고, Solaris에서는 컴파일시 공유 라이브러리를 지정했다면 그 라이브러리 이름은 모두 로드할 때를 대비하여 결과물 파일 안에 저장하게 된다. 그러나 AIX 4.2 와 이후 버전에서는 `-brtl` 옵션을 사용하면 명령행에서 지정한 모든 공유 오브젝트(아카이브 멤버는 제외)를 결과물 파일 안에 저장하게 되서 Solaris와 동일한 경우로 만들 수 있다.

Solaris에서 공유 오브젝트를 로드하게 되면 모든 심볼참조는 실시간 링커가 수행하게 되는데, 프로세스에서 현재 로드한 모든 모듈을 찾는다. 대부분의 경우, 성능을 향상시키기 위해 함수는 실제 그 함수를 처음 수행할 때까지 심볼참조를 미룬다.(변수참조는 처음 로드할 때 수행한다.) 이렇게 되면 함수 심볼을 참조하는 모듈을 로드하고 난 다음에야 실제 함수 정의를 로드할 수 있다. AIX에서는 실시간 링커를 사용하는 경우라고 하더라도, 모든 심볼(“deferred”로 지정한 심볼은 제외)은 로드할 때 심볼 참조가 수행되어야 한다. 실시간 링커에서는 특정 심볼이 “deferred”로 지정되어 있다면, 그 심볼은 로드시 참조되지 않은 상태로 그냥 둔다. deferred 심볼을 참조되는 상태로 만드는 역할은 전적으로 어플리케이션 프로그래머의 책임이다. 어떤 때는 참조될 모듈이 로드될 때 비로소 프로세스에서 심볼에 대한 정의를 가져가는 경우도 있다. AIX에서 링커의 이러한 동작으로 인해, 어플리케이션

프로그래머는 좀더 좋은 성능을 내기 위해 프로그램내에서 인터페이스 정의를 정확하게 해줘야 하는 부담이 있다.

AIX외의 다른 플랫폼에서는 특정 심볼을 지정한 라이브러리로 반드시 연관시키도록 할 수 없기 때문에, 라이브러리를 지정하는 순서가 매우 중요하다. rpc를 예로 들어보자. 만약 스트림 기능을 사용하려고 한다면 라이브러리 순서는 libnsl 다음 libc를 사용해야 하고, 소켓기능을 사용하려고 한다면 libc 다음 libnsl을 사용해야 한다.

예를 들어

1. 모듈 A에 함수 X 에 대한 정의가 들어있다.
2. foo 라이브러리에는 함수 X 에 대한 다른 정의가 들어있다.
3. foo 라이브러리의 Y 함수는 함수 X를 호출한다.

만약 이 모듈과 라이브러리를 모두 사용하는 어플리케이션이 A, foo 순서로 링크순서를 지정했다면 다음과 같은 방식으로 심볼참조가 일어난다.

AIX : 함수 Y는 foo 라이브러리에서 정의한 X 를 호출한다.
Solaris : 함수 Y는 모듈 A 에서 정의한 X 를 호출한다.

만약 어플리케이션을 링크할 때 명령행에서 foo, A 순서로 링크 순서를 바꾼다면, AIX의 결과는 바뀌지 않지만, Solaris에서는 함수 Y가 foo 라이브러리에서 정의한 X 를 호출하게 된다.

AIX만의 고유 특징으로 실행파일을 다시 링크할 수 있는 기능이 있다. 즉 ld 명령어를 사용해서 만든 출력파일을 다시 입력파일로 사용할 수 있다. 이러한 기능을 사용하면 새로운 오브젝트 파일(.o 파일)을 기존의 어플리케이션과 링크하여 어플리케이션을 완전히 새로 만들지 않고도 어플리케이션 내의 코드를 수정할 수 있다. 따라서 어플리케이션을 개발할 때 상당히 많은 시간을 절약할 수 있다. AIX 링커와 로더의 기능, 링커 옵션, 예제등등에 대해 좀더 자세히 알아보고 싶으면 “AIX Linking and Loading Mechanisms” 문서를 참고하기 바란다.

라이브러리 검색 경로 생성과 사용방법에서의 AIX와 Solaris의 차이점

이제 Solaris환경에서 링크할 때와 실행시간 중(로더)에 라이브러리를 찾는 경로를 어떻게 다루는지 살펴보자.

-L	–l 플랙으로 지정한 라이브러리를 찾을 때 -L 링커 플랙에서 지정한 경로를 사용한다. 이 옵션은 링커에서 사용하게 된다.
LD_LIBRARY_PATH #1 과 #2	이 환경변수는 libpath#1[;libpath#2]와 같은 모양으로 사용한다. 이 정보는 링커, 로더 둘 다 사용한다.
-R	이 옵션은 링커에게 직접 명령행에서 줄 수 있는 플랙이다. 링커는 ELF 오브젝트의

	동적 섹션에 관련된 라이브러리 탐색경로를 저장한다.
LD_RUN_PATH	링커는 이 환경변수 값(라이브러리 탐색 경로)을 ELF 오브젝트의 동적 섹션에 저장한다.

다음은 Solaris 환경에 적용되는 규칙이다.

1. 디폴트 탐색 경로는 `/usr/lib` 오직 한 개만 존재한다. 그리고 이 값은 가장 마지막에 사용된다.
2. 이 외의 모든 탐색 경로는 명시적으로 탐색경로 값에 추가해야만 한다.
3. `-L`과 `LD_LIBRARY_PATH`값은 서로 섞어서 사용된다.
   예를 들면
   `-L` path3
   `LD_LIBRARY_PATH` 값은 "path1;path2" 로 되어 있을 때
   `-l` library 를 찾는 실제 탐색 경로는 path1:path3:path2:/usr/lib 순서가 된다.
4. `-R` [경로] 값은 `RPATH` 정보로 ELF 오브젝트의 동적섹션에 저장된다.
5. `-L` 과 `LD_LIBRARY_PATH` 로 지정된 경로는 `-l` [라이브러리] 의 라이브러리를 찾는데 사용한다. `-R` [경로] 나 ELF 에 `RPATH`로 지정된 경로 값은 실시간에 로더에서 사용하는 값이다.
6. 만약 `RPATH`값만으로 실시간 오브젝트를 로드할 수 없다면 보충정보로 `LD_LIBRARY_PATH` 값을 사용한다.

이제 AIX 환경에서 라이브러리를 탐색경로를 다루는 방법을 살펴보자.

-L	–l 플랙으로 지정한 라이브러리를 찾을 때 -L 링커 플랙에서 지정한 경로를 사용한다. 이 옵션은 링커에서 사용하게 된다.
-blibpath	path값은 로더섹션의 디폴트 라이브러리 탐색 경로(0번째 인덱스 값)로 저장된다. 이 작업은 링커에서 수행한다.
-bnolibpath	경로 값에 대한 정보를 제공하지 않으며 링커에 대해 단순히 플랙으로 동작한다.
#!path	반입(import)항목의 오브젝트나 공유 라이브러리와 연관된 경로값. 로더섹션에 고정된 경로 값으로 기록된다.
LIBPATH	라이브러리를 검색하는데 사용되지는 않지만, 특정 환경에서 로더섹션에 값이 기록된다.

다음은 AIX 환경에 적용되는 규칙이다.

- 명령행에서 직접 링크시키는 명령을 수행하면, "`-l`" 로 지정한 라이브러리는 "`-L`"로 지정한 경로상에서 찾는다.
- 라이브러리가 존재하는 경로이며 `-L` 로 명시가 되어 있을 때 그 경로값은 로더섹션의 0번째 인덱스 라인에 `/usr/lib`, `/lib` 다음에 기록된다. `LIBPATH` 환경변수 값은 여기에 기록되지 않는다.

- 반입(import)해야 할 항목이 있다면, 반입파일에 #! 매직문자 다음 경로값과 해당하는 라이브러리 이름이 나오게 된다. 이 값은 로더섹션에 기록된다. 특정 라이브러리의 경우에는 경로값이 고정되어 있는 경우가 있는데, 이 경우 경로값은 절대경로이며 실시간 동안에는 바꿀 수 없다.
- ld 명령어를 직접 사용할 때 명령행에서 –blibpath:pathname 옵션을 사용하면 경로값은 인덱스 0값에 기록된다. 이 값은 로더섹션에 기록되는 유일한 라이브러리 경로값이며 /usr/lib:/lib 값조차도 사용되지 않는다. -l 로 지정한 라이브러리를 찾을 때는 -L 로 지정한 경로값을 사용하여 라이브러리를 찾지만 경로값 자체는 로더섹션에 기록되지 않는다. 명령행에서 -L 이나 -blibpath 값을 모두 사용하지 않았다면, LIBPATH 환경변수에 들어있는 값이 로더섹션의 인덱스 0에 기록된다. 그리고 -bnolibpath 를 지정하면 LIBPATH 값만이 기록된다.

라이브러리 검색 경로에서의 AIX와 Solaris의 차이점

1. Solaris와 AIX 모두 오브젝트의 실시간 탐색 경로 정보를 저장하기 위해 링커를 사용한다.
2. -l 로 지정한 라이브러리를 찾을 때 Solaris의 링커는 LD_LIBRARY_PATH #1, -L path, LD_LIBRARY_PATH #2 순서대로 찾으며, AIX는 -L 로 지정한 경로만을 사용한다. 여기에서 AIX의 LIBPATH 값은 사용하지 않는다.
3. Solaris는 ELF 파일의 동적섹션에 -R (RPATH) 와 LD_RUN_PATH 환경변수 값을 저장한다. AIX는 실시간 경로에 대한 정보를 XCOFF 파일의 로더섹션에 저장한다.
   a) 반입파일에 적혀있는 고정된 경로값과 이에 해당하는 라이브러리 이름에 대한 정보를 저장한다.
   b) -blibpath 가 없다면, 실제 -L 값이 저장된다.
   c) -blibpath 값이 있다면, 해당되는 경로값이 저장된다. 이 경우는 Solaris의 -R 플랙을 사용하는 것과 동일하지만, AIX에는 정확히 LD_RUN_PATH에 해당하는 변수가 없다.
   d) 명령행에서 -bnolibpath 를 사용한 경우 LIBPATH 값은 기록이 되고 -L 로 지정한 경로값은 없어진다. 이 경우는 Solaris의 LD_RUN_PATH 와 비슷하다.
4. 실시간 오브젝트에 포함된 실시간 경로 정보가 경로를 결정하는데 충분하지 못한 경우:
   a) Solaris는 LD_LIBRARY_PATH 값을 사용해서 나머지 경로를 결정한다.
   b) AIX는 같은 목적으로 LIBPATH 값을 사용한다.

메모리 세그먼트

AIX에서는 하드웨어적으로 0x00000000000000000000에서 0xFFFFFFFFFFFFFFFFFFFF 까지의 연속된 가상 주소 공간을 제공하며, 접근 가능한 전체 주소공간은 1 trillion 테라바이트를 넘는다. 메모리에 접근하는 명령어는 64bit 주소를 사용하는데, 이중 36bit 는 세그먼트 레지스터를 선택하는데 사용하고 나머지 28bit로 세그먼트 내 옵셋을 결정한다. 이러한 어드레싱 방식으로 256메가바이트의 세그먼트를 6천4백만개 이상 접근할 수 있다. 각각의 세그먼트 레지스터는 52bit 세그먼트 ID를 가지는데 28bit 옵셋값을 붙여서 가상메모리 주소값을 결정한다. 이

80bit 가상주소 값으로 전체 가상 메모리 공간의 주소를 결정하게 된다. 프로세스 공간은 64bit 주소를 사용하기 때문에 포인터도 64bit 가 된다. 그러나 각각의 프로세스나 인터럽트 핸들러는 세그먼트 레지스터에 해당 세그먼트 ID 가 들어있는 경우 시스템의 가상메모리 공간(세그먼트)을 참조할 수 있다.

시스템에서 수행되는 64bit 어플리케이션 프로그램은 다음과 같이 두개의 세그먼트로 나누어지는 주소공간을 갖게 된다.

메모리 주소 범위	설명
0x0000 0000 0000 0000 ® 0x0000 0000 0fff ffff	커널 포함 영역
0x0000 0000 d000 0000 ® 0x0000 0000 dfff ffff	32bit 공유 라이브러리 텍스트 포함영역(64bit 어플리케이션에서는 접근할 수 없음)
0x0000 0000 e000 0000 ® 0x0000 0000 efff ffff	32bit 어플리케이션에서 사용할 수 있는 공유 메모리 세그먼트
0x0000 0000 f000 0000 ® 0x0000 0000 0fff ffff	32bit 공유 라이브러리 데이터 (32bit 어플리케이션에서는 접근할 수 없음)
0x0000 0001 0000 0000 ® 0x07ff ffff ffff ffff	어플리케이션 프로그램 텍스트와 데이터를 포함하는 영역. 이외에도 공유 메모리나 mmap 서비스에도 사용된다.
0x0800 0000 0000 0000 ® 0x08ff ffff ffff ffff	프로그램 내에서 로드한 64bit 모듈
0x0900 0000 0000 0000 ® 0x09ff ffff ffff ffff	64bit 공유 라이브러리 텍스트와 데이터
0x0f00 0000 0000 0000 ® 0x0fff ffff ffff ffff	64bit 어플리케이션 스택

테이블 1.3 프로그램 주소 공간

AIX에서 64bit 어플리케이션은 0x0000000100000000 에 로드된다. 그런데 커널도 메모리의 시작부분인 0x0(low memory)부터 매핑된다. 따라서 32bit 포인터를 사용하면 64bit 어플리케이션 주소공간을 전혀 참조할 수 없지만, 반면 커널 세그먼트(low memory에 존재)를 읽는 것은 가능하다.

32bit 주소 공간

다음 테이블에서 32bit 주소공간이 나누어져 있는 상황을 알 수 있다. 32bit 어플리케이션 프로그램은 7개의 세그먼트로 구분된 주소 공간을 가진다.

메모리 주소 범위	설명
0x00000000 - 0x0fffffff	커널 포함 영역
0x10000000 - 0x1fffffff	어플리케이션 프로그램 텍스트 포함
0x20000000 - 0x2fffffff	어플리케이션 프로그램 데이터와 어플리케이션 스택 포함
0x30000000 - 0xcfffffff	공유 메모리나 mmap서비스가 사용할 수 있도록 예약

0xd0000000 - 0xdfffffff	공유 라이브러리 텍스트 포함
0xe0000000 - 0xefffffff	커널 데이터 영역. 이 외에도 공유 메모리나 mmap서비스가 사용할 수 있도록 예약
0xf0000000 - 0x0fffffff	어플리케이션 공유 라이브러리 데이터 포함

AIX 메모리 모델에 대한 더 자세한 정보를 알아보려면 AIX 온라인 문서 http://publib.boulder.ibm.com/cgi-bin/ds_form 에서 “Program Address Space Overview” 나 “Understanding Memory Mapping” 로 검색해보기 바란다.

런타임 이슈 - 32bit 대 64bit 커널

AIX 4.3.3 에서 64bit 어플리케이션을 수행하려면 `bos.64bit` 파일셑을 설치해야 한다.(`lslpp –L|grep bos.64bit` 명령어를 사용해서 이 파일셑이 설치되어 있는지 여부를 알 수 있다. 만약 설치되어 있지 않다면 AIX 설치 CD 에서 설치해야 한다.)

AIX5L은 32bit 나 64bit 커널 중 하나를 선택해서 부팅할 수 있다. AIX5L에서는 32bit, 64bit 커널여부에 상관없이 모든 64bit 어플리케이션을 수행할 수 있으며 32bit 어플리케이션도 마찬가지이다. 그러나 시스템에서 사용하는 대부분의 중요 어플리케이션이 64bit 라면 하드웨어의 성능을 최대한으로 이용하기 위해 커널도 64bit 로 부팅하는 편이 좋다. 64bit 커널은 `bos.mp64` 파일셑에 들어있으며, 처음 AIX를 설치할 때 “Installation and Settings”화면의 “Advanced Options”에서 “Enable 64-bit Kernel and JFS2”를 선택하면 이 파일셑은 자동으로 설치된다. 이 경우 디폴트 64bit 커널로 부팅된다.

이 외의 경우에는 32bit 커널이 디폴트 커널이 된다. 일단 설치한 후라고 해도 다음 명령어를 사용하면 64bit 커널모드로 바꿀 수 있다.

```
> ln -sf /usr/lib/boot/unix_xx /unix
> ln -sf /usr/lib/boot/unix_xx /usr/lib/boot/unix
> bosboot -ad /dev/ipldevice
> shutdown -r
```

여기에서 unix_xx 를 해당 사항으로 바꾼다.

- unix_up - 32 bit uni kernel
- unix_mp - 32 bit mp kernel
- unix_64 - 64 bit mp kernel

AIX 4.3에서 컴파일한 32-bit, 64-bit 어플리케이션을 AIX 5.1에서 수행하기

AIX 4.3에서 컴파일한 32bit 어플리케이션은 AIX 5.1과 바이너리 호환성을 가지며 AIX 5.1 상에서 그대로 수행할 수 있고 다시 컴파일하지 않아도 된다. 그러나 AIX 4.3에서 컴파일한 64bit 어플리케이션을 AIX 5.1에서 수행하려면 5.1상에서 다시 컴파일해야 한다.

성능 튜닝

AIX와 Solaris의 가장 큰 차이점이라면 AIX는 사용자나 시스템 관리자가 커널환경파일을 수정하고 다시 커널을 빌드할 필요가 없다는데 있다. AIX에서 튜닝 가능한 파라미터는 모두 `smit` 명령어(`smitty chgsys`)나 `vmtune`, `schedtune`등 `/usr/samples/kernel` 에 있는 명령어를 통해서만 수정할 수 있다.

성능 관련한 문서는 Performance Management Guide 를 참고하시오.

쓰레드 프로그래밍

Solaris 쓰레드 프로그래밍 인터페이스(ex)`thr_create()`, `mutex_lock()`, `cond_signal()`) 를 사용하는 경우 일반적으로 포팅이 매우 어렵다. 이러한 인터페이스를 쓰는 프로그램을 AIX로 포팅할 때 수정해야 할 사항이 많다.

POSIX 쓰레드에서 지원되지 않는 Solaris 쓰레드 함수 종류는 다음과 같다.

- 데몬 쓰레드(`thr_create`)
- 쓰레드 연결하기 (Solaris에서는 tid가 0인 쓰레드는 연결할 수 있다)
- 쓰레드 서스펜드, 쓰레드 계속실행

Solaris는 Solaris 쓰레드 API 외에도 POSIX 쓰레드 API를 지원하므로, Solaris에서 사용하는 쓰레드 어플리케이션을 쉽게 포팅하려면 Solaris 쓰레드 API 부분을 모두 POSIX 쓰레드 API를 사용하도록 바꿔야 한다. 그러나 어떤 Solaris 쓰레드 API의 기능은 POSIX 쓰레드 API로 구현할 수 없는 경우가 있으므로, 이런 경우 변환 작업이 매우 어렵게 된다. 어플리케이션에 따라 몇 줄 정도만 고치면 되는 경우도 있고, 아예 전체 설계부터 다시 해야 하는 경우도 있다.

어플리케이션이 Solaris 쓰레드 API를 사용하고 있는지 POSIX 쓰레드 API를 사용하고 있는지는 쓰레드 라이브러리 이름으로 구분할 수 있다. 어플리케이션에 링크한 라이브러리가 "`-lthreads`" 면 Solaris 쓰레드 라이브러리(STL)을 사용한 경우이고, "`-lpthreads`"이면 POSIX 쓰레드 라이브러리를 사용한 경우이다.

다음 테이블은 Solaris 쓰레드 API와 POSIX 쓰레드 API를 비교하고 있다.

POSIX 쓰레드 API	Solaris 쓰레드 API	기능
pthread_create()	thr_create()	새로운 쓰레드를 생성한다.
pthread_exit()	thr_exit()	쓰레드의 수행을 중지한다.
pthread_getschedparam()	thr_getprio()	쓰레드의 우선순위 파라미터 값을 얻어온다.
pthread_getspecific()	thr_getspecific()	쓰레드의 특정값과 키값을 연결한다.
pthread_join()	thr_join()	다른 쓰레드가 작업을 마칠 때까지 쓰레드를 실행 중지시킨다.
pthread_key_create()	thr_keycreate()	쓰레드의 데이터위치에 따른 키를 생성한다.
pthread_kill()	thr_kill()	시그널을 다른 쓰레드에게 보낸다.
pthread_self()	thr_self()	호출하는 프로세스의 쓰레드 ID를 반환한다.
pthread_setschedparam()	thr_setprio()	쓰레드의 우선순위 파라미터 값을 바꾼다.
pthread_setspecific()	thr_setspecific()	쓰레드의 특정값과 키값을 연결한다.
pthread_sigmask()	thr_sigsetmask()	쓰레드의 시그널마스크 값을 바꾸거나 검사한다.
sched_yield()	thr_yield()	현재 수행되는 쓰레드를 다른 쓰레드로 바꾼다.
pthread_setconcurrency()	thr_setconcurrency()	쓰레드 콘커런시 레벨을 지정한다.
pthread_getconcurrency()	thr_getconcurrency()	쓰레드 콘커런시 레벨을 얻어온다.
pthread_setspecific()	thr_setspecific()	쓰레드 데이터 키 값을 지정한다.
pthread_getspecific()	thr_getspecific()	쓰레드 데이터 키 값을 얻어온다.
지원 안됨	thr_suspend	지정한 쓰레드의 수행을 일시적으로 중지시킨다.
지원 안됨	thr_continue	일시적으로 중지시킨 쓰레드를 다시 시작한다.

테이블 1.4 : 쓰레드 API

POSIX 쓰레드에는 A쓰레드가 다른 B 쓰레드의 협력없이 B 쓰레드의 수행을 중지시킬 수 있는 메커니즘이 없다. 쓰레드를 수행 중지시켰다가 다시 시작하도록 하는 메커니즘을 구현하려면, 우선 B 쓰레드가 전역변수를 주기적으로 검사해서 suspend 요청이 들어왔는지 보고 만약 suspend 요청이 들어왔다면 스스로를 수행 중지시키는 방식으로 해야 한다. 다음 다시 수행하도록 하려면 다른 쓰레드가 이 B 쓰레드에게 resume 시그널을 보내는 식으로 해서 구현하는 수밖에 없다.

POSIX 쓰레드 라이브러리 대 Solaris 쓰레드 라이브러리

Solaris는 POSIX 1003.1c 와 Solaris 고유의 세마포어 API 모두를 지원한다. 다음 테이블 1.5 는 POSIX 1003.1c API와 SUN API 를 비교한다. 다음 함수는 Solaris에서는 **libthread**에 정의되어 있고, AIX에서는 **libpthread**에 정의되어 있다.

POSIX 라이브러리(libpthread)	Solaris 쓰레드 라이브러리	기능

	(libthread)	
pthread_mutex_destroy()	mutex_destroy()	mutex 오브젝트와 관련된 상태변수를 제거하거나 해제한다.
pthread_mutex_init()	mutex_init()	mutex 변수를 초기화한다.
pthread_mutex_lock()	mutex_lock()	nutex 오브젝트를 사용하지 않을 때까지 mutex 오브젝트에 락을 건다.
pthread_mutex_unlock()	mutex_unlock()	mutex 오브젝트를 해제한다.
pthread_cond_broadcast()	cond_broadcast()	condition 변수가 바뀌기를 기다리는 모든 쓰레드를 해제한다.
pthread_cond_destroy()	cond_destroy()	condition 변수와 관련된 모든 상태변수를 제거한다.
pthread_cond_init()	cond_init()	condition 변수를 초기화한다.
pthread_cond_signal()	cond_signal()	condition 변수가 바뀌기를 기다리는 다음 쓰레드를 해제한다.
pthread_cond_wait()	cond_wait()	모든 condition 변수를 블락시킨 후 끝에 한꺼번에 해제한다.
pthread_rwlock_init()	rwlock_init()	read/write 락을 초기화한다.
pthread_rwlock_destroy()	rwlock_destroy()	read/write 락에 락을 설정한다.
pthread_rwlock_rdlock()	rw_rdlock()	read/write 락에 read락을 설정한다.
pthread_rwlock_wrlock()	rw_wrlock()	read/write 락에 write락을 설정한다.
pthread_rwlock_unlock()	rw_unlock()	read/write 락을 해제한다.
pthread_rwlock_tryrdlock()	rw_tryrdlock()	NON-블락킹 read/write 락으로 read락을 설정한다.
pthread_rwlock_trywrlock()	rw_trywrlock()	NON-블락킹 read/write 락으로 write락을 설정한다.

테이블 1.5 : 쓰레드 라이브러리 함수

POSIX 쓰레드 대신 Solaris의 쓰레드 API를 사용한 어플리케이션이라면, `fork()`함수는 부모 프로세스의 모든 LWP(Light Weight Process)와 쓰레드를 모두 복제한다. `fork1()`함수를 사용해야 호출한 쓰레드만을 복제한다.

그러나 POSIX 쓰레드 API를 사용한 어플리케이션에서는, `fork()`를 호출하면 `fork1()`을 호출한 것과 같은 효과가 발생해서 호출한 쓰레드만을 복제하게 된다. POSIX 쓰레드의 경우에는 Solaris에서 처럼 모든 쓰레드와

LWP를 복제하는 함수는 없다.

만약 Solaris상에서 프로그램 링크 시에 두 라이브러리(`-lthread` 와 `-lpthread`)를 모두 사용한다면, 실시간에는 POSIX에서의 `fork()`의미로 사용된다.

시그널

AIX5L 에서는 POSIX, BSD, System V 스타일의 함수를 모두 제공하므로, 포팅상 시그널과 연관된 코드는 쉽게 처리할 수 있다. AIX및 다른 벤더의 UNIX에서 지원하는 시그널의 자세한 목록은 “IBM Redbook, AIX 5L Porting Guide”의 7장을 참고하시오.

포팅 이슈

이 섹션에서는 Solaris에서 AIX로 포팅하는 중에 발생하기 쉬운 여러 가지 이슈에 대해 다루어 보겠다.

컴파일러 에러

Sun의 Forte 컴파일러와 IBM의 VisualAge for C/C++ 컴파일러를 비교해보면, IBM측이 좀 더 비표준적인 생성자에 대해 엄격하게 규칙을 적용하는 편이다. 따라서 Sun측에서는 발생하지 않는 컴파일 에러 문제가 IBM측에서는 발생할 수도 있다. 흔한 에러 및 해결방안에 대해 아래에서 기재하고 있다.

표준 C++ 헤더파일(.h 확장자가 있는 경우와 없는 경우)

새로운 C++ 표준에 의하면, 표준 C++ 헤더를 사용할 때는 .h 확장자 없이 그냥 사용해야 한다. 따라서 기존에 "`#include <iostream.h>`” 와 같이 사용하던 구문을 "`#include <iostream>`"로 사용해야 한다. Solaris의 Forte C++ 컴파일러에서는 C++ 표준헤더를 쓸 때 .h 확장자를 써도 되고 쓰지 않아도 되도록 섞어서 사용한다.(실제로 Forte C++ 컴파일러에서는 .h 확장자가 있는 표준헤더 파일 안에 .h 확장자가 없는 표준파일 및 using 구문을 포함하고 있다. 이런 경우 컴파일러 업그레이드가 좀 더 쉽다.) 그러나 AIX의 VisualAge C++ 컴파일러는 이렇게 섞어서 사용하는 방식을 허용하지 않는다. 따라서 AIX용으로 사용하려면 C++ 표준 헤더파일의 .h 확장자를 제거해야 한다.

네임스페이스

AIX에서 네임스페이스는 매우 엄격하게 처리하며 STL(Standard Template Library) 에서는 더더욱 엄격하게 처리된다. “`using namespace std`”, “`using std::string`”, “`using std::vector`” 등등의 구문은 헤더파일에 제대로 써주지 않으면 컴파일 에러가 발생한다.

콘테이너

AIX에서는 map이나 vector같은 콘테이너가 const 타입이 되는 것을 허용하지 않는다. 특히 "`std::map< const std::string, Foo>`"와 같은 구문은 AIX에서 컴파일이 안 된다. map의 key값은 당연히 변경할 수 없는 값이므로 const 구문을 사용하는 것은 의미가 없다. (key,value)값의 쌍으로 값이 반환될 때 key 타입은 const가 되지만, 그렇다고 일부러 map 템플리트에서 const로 선언하면 컴파일이 안 된다.

캐스팅

AIX 컴파일러에서는 캐스팅을 엄격하게 적용한다. 예를 들어 "`const void *`"를 const type이 아닌 포인터로 캐스팅을 할 수 없다.

fstream

fstream 오브젝트로부터 파일 디스크립터를 얻어내기 위해 Solaris에서는 `fstream::rdbuf().fd()`메쏘드를 사용한다. 그러나 AIX의 `basic_filebuf` 클래스에서는 `fd()` 메쏘드를 지원하지 않는다.

```
tout = new ofstream(_filename, ios::app, 0666);
if(tout->good()) {
        dup2(tout->rdbuf()->fd(), 1);
}
```

AIX에서는 아래와 같이 구현할 수 있는데, 우선 `filebuf` 오브젝트를 생성한 다음 `mybuf`로 이름 짓는다. 다음 멤버함수인 `str.rdbuf(mybuf)`를 호출해서 버퍼와 오브젝트 str의 ifstream 혹은 ofstream을 연결시킨다.

```
#ifdef SUN5
tout = new ofstream(dup(1)); //attach stdout to it...
#else // defined AIX
tout = new ofstream();
FILE *fp = fdopen(dup(1), "a+");
filebuf *mybuf = new filebuf(fp);
((basic_ios<char, char_traits<char> > *)tout)->rdbuf(mybuf);
#endif
```

```
if(tout->good()) {
#ifdef SUN5
dup2(tout->rdbuf()->fd(), 1);
#else // defined AIX
dup2(fd, 1);
#endif
```

fdopen

fdopen은 openx, dup, create, pipe 등등의 서브루틴에서 얻은 파일 디스트립터와 stream을 연결한다. dup같은 서브루틴은 파일을 open하지만, FILE 구조체에 포인터를 반환하지는 않는다.대부분의 표준 I/O 서브루틴은 FILE구조체에 대한 포인터를 반환하게 되어 있다. AIX에서는 fdopen이 dup서브루틴에서 반환한 파일 디스크립터를 사용하게 되면, 동일한 권한으로 중복된 파일 디스크립터가 파일을 open하게 된다. 예를 들어, 원래 파일을 읽기 전용(read-only)로 생성했다면, 스트림도 읽기 전용 권한만 가지게 된다. 이러한 사항은 fdopen 서브루틴의 'type' 파라미터에서 지정할 수 있다.

```
FILE *fdopen (FileDescriptor, Type)
int FileDescriptor;
const char *Type;
```

fdopen 서브루틴의 type파라미터는 스트림의 모드를 지정하는데, 예를 들어'r'은 읽기 전용으로 파일을 open하라는 뜻이고 'a'는 파일끝부분에 write할 수 있도록 파일을 open하라는 뜻이다. fdopen 서브루틴에서 type 파라미터로 지정한 모드 값은 원래 open한 파일의 모드 값과 일치해야 한다.

char constant 참조

다음 코드는 Sun Forte 컴파일러로는 컴파일할 수 있지만, AIX에서는 안 된다.

```
#include <iostream.h>
class test
{
public:
// Using a constant ref. to a variable whose value can be changed
       void test1 ( char & const newintrlEvent){
              cout << newintrlEvent <<endl;
```

```
        }
};
int main()
{
        test t;
        char * const ptr="ABC";
        t.test1(*ptr);
        return 0;
}
```

다음 코드는 AIX에서 잘 컴파일된다.

```
#include <iostream>
class test
{
public:
// Using a constant ref. to a variable whose value can be changed
        void test1 ( char const & newintrlEvent){
                cout << newintrlEvent << endl;
        }
};
int main()
{
        test t;
        char * const ptr="ABC";
        t.test1(*ptr);
        return 0;
}
```

다음은 두 코드의 차이점이다. 잘 살펴보자.

Sun Forte	AIX
#include <iostream.h> class test{ public:	#include <iostream.h> class test{ public:

// Using a constant ref. to a
// variable whose value can be
// changed
	// Using a constant ref. to a
	// variable whose value can be
	// changed

void test1(char& const nEvt){	void test1(**char const & nEvt**){

cout << nEvt << endl;
}
};
int main()
{
test t;
char * const ptr="ABC";
t.test1(*ptr);
return 0;
}
	cout << nEvt << endl;
	}
	};
	int main()
	{
	test t;
	char * const ptr="ABC";
	t.test1(*ptr);
	return 0;
	}

`char& const` 와 `char const &` 는 다르다. `char& const`는 constant 값에 대한 레퍼런스이고, `char const &` 는 레퍼런스 값이 아닌 레퍼런스 자체가 constant 값이 된다.

const 참조

다음 코드는 Solaris에서는 깨끗하게 컴파일되지만, AIX에서는 에러가 발생한다.

```
class T {
       public:
              T(int);
};
class U {
       public:
              U(T& k=T(0));
};
```

다음은 AIX에서 발생하는 에러 메시지이다.

```
"test.cpp", line 10.16: 1540-1280 (S) An rvalue of type "T" cannot be converted
to "T &".
```

```
"test.cpp", line 10.16: 1540-1290 (I) An rvalue cannot be converted to a
reference to a non-const type.
```

위 코드를 AIX에서 제대로 컴파일 하려면 파라미터에 const를 붙여야 한다. 다음은 Solaris와 AIX 코드를 비교하고 있다.

Solaris	AIX

class T {
public:
T(int);
};
class U {
public:
U(T& k=T(0));
};
	class T {
	public:
	T(int);
	};
	class U {
	public:
	U(const T& k=T(0));
	};

파라미터 값을 const로 지정하지 않으면 class 생성자가 레퍼런스값을 갱신할 가능성이 있다. 레퍼런스값은 일시적인 값을 가리키기 때문에 업데이트되면 값이 바뀌거나 없어질 수 있다. AIX에서 이 코드를 제대로 컴파일하려면 코드를 `U(const T&k=T(0))`로 변경해야 한다. 그리고 이렇게 하는 게 더 C++답게 사용하는 방식으로 볼 수 있다.

참조자로 반환된 오브젝트 저장하기

다음 코드(아래 테이블의 왼쪽 칼럼 코드)는 Solaris에서는 잘 컴파일되지만 AIX에서는 에러메시지가 발생한다. 해결방법 2가지를 테이블 오른쪽 컬럼에 실어놓았다.

Solaris	AIX

class foo {
public:
foo();
};
 	
foo getFoo( ) {
foo myFoo;
return myFoo;
}
	class foo {
	public:
	foo();
	};
	
	foo getFoo( ) {
	foo myFoo;
	return myFoo;
	}

`void myFunc( ) {`	`void myFunc( ) {`
`foo& myFoo = getFoo();`	*Solution 1:* `const foo & myFoo = getFoo();`
	Solution 2: `foo tmpFoo =getFoo();` `foo& myFoo = tmpFoo;`
`}`	`}`

C++ 표준에 의하면, `getFoo()`같은 표현식은 const가 아닌 레퍼런스와 바인드시킬 수 없다. 따라서 AIX의 VisualAge C++ 컴파일러가 좀더 표준안을 엄격하게 적용하고 있다.

Copy constructors, equals operator, comparison operator

VisualAge에서는 어떤 경우 copy 생성자, equals 연산자, 비교 연산자 파라미터값을 const값으로 요구한다. 따라서 가능한 에러를 피하려면 아예 모든 파라미터 값을 const로 하는 편이 좋다. 다음은 예제이다.

Solaris	AIX

`Foo(Foo&)`
`Foo & operator = (Foo &)`
`Boolean operator == (Foo &,Foo &)`
	`Foo(const Foo&)`
	`Foo & operator = (const Foo &)`
	`Boolean operator == (const Foo &, const Foo &)`

메쏘드 네이밍

VisualAge C++에서 메쏘드 이름은 클래스 이름과 동일한 이름을 사용할 수 없다. 아래 테이블의 3번째 줄을 참고하시오.

Solaris	AIX

`class Foo;`
`class FooNext {`
`  Foo *Foo;`
`};`
	`class Foo;`
	`class FooNext {`
	`  **Foo *fooPtr;**`
	`};`

friend 사용법

VisualAge에서는 `friend` 키워드를 붙이는 것만으로 클래스를 생성하지는 않는다. `friend`로 만들기 이전에 우선 클래스 선언부터 해야 한다. 다음 예제를 보시오.

Solaris	AIX
`class Foo{`	**`class FooFriend;`**

friend class FooFriend;
};
	class Foo {
	friend class FooFriend;
	};

static friend 사용법

Solaris의 Forte C++ 컴파일러는 friend함수가 static이 되는 것을 허용하지만 AIX VisualAge C++ 컴파일러에서는 허용하지 않는다. 따라서 friend함수를 static으로 지정하지 않아야 컴파일 에러를 줄일 수 있다.

class qualifier

VisualAge는 클래스 정의 안에 class qualifier :: 을 허용하지 않는다.

Solaris Forte	AIX VisualAge

class FooNext {
void FooNext::getFoo();
};
	class FooNext {
	void getFoo();
	};

#endif 다음 텍스트

VisualAge에서는 #endif 다음에 주석문을 제외하고 다른 텍스트가 나올 수 없다.

Solaris Forte	AIX VisualAge
#endif AIX	#endif // AIX

링크드 리스트

다음 코드는 Solaris에서는 잘 컴파일되지만, AIX에서는 컴파일 에러가 발생한다.

```
/* Type definition for a linked list and linked list pointer. */
class LinkList : public Node {
   int _listlength; /* no. of elements in linked list */
   int _itemsize; /* size of the data _item (see _NODE) */
  public :
   Node *_head; /* pointer to _head of linked list */
   Node *_tail; /* pointer to _tail of linked list */
   Node *_current; /* pointer to _current node */
   -
   -
```

```
}

LinkList operator++(){
   if( _current->_next )
      _current = _current->_next;
   else
      _current = NULL;
   return *this;
}


_
_
LinkList LinkList::operator=(LinkList &from_list)
{
   char *from_item,
       *to_item;
   LinkList _list(sizeof(this->_head));
-
-
   from_list++;
-
}
```

from_list++; 에서의 에러메시지는 다음과 같다.

```
1540-0218 (S) The call does not match any parameter list for "operator++".
1540-0215 (I) The wrong number of arguments have been specified for
"LinkList::operator++()".
1540-1283 (I) "LinkList::operator++()" is not a viable candidate.
```

이 코드를 AIX에서 제대로 컴파일되게 하려면

```
LinkList operator++() {
```

를

```
LinkList operator++(int ){
```

로 수정해야 한다.

제대로 컴파일 안 되는 원인은 다음과 같다. 예를 들어 `++b` 와 같은 코드를 만나면 컴파일러는 `B::operator++( )`로 해석하고, 반대로 `b++` 와 같은 코드를 만나면 `B::operator++(int)` 로 해석하기 때문이다.

C++ 스타일 주석문

C 소스 파일에서 C++ 스타일의 주석문을 사용하려면 컴파일할 때 `-qcpluscmt` 옵션을 사용한다.

ENONET 에러

Solaris에서는 ENONET("Machine is not on the network") 에러 심볼을 지원하지만, AIX는 지원하지 않는다. AIX에서 이와 유사한 심볼은 ENETUNREACH ("Network is unreachable")가 해당된다.

MAXHOSTNAMELEN 매크로

Solaris에서는 `MAXHOSTNAMELEN` 매크로가 `netdb.h` 에 정의되어 있고, AIX에서는 `sys/param.h` 에 정의되어 있다.

시스템 콜 차이

Solaris용 어플리케이션에서 사용하는 몇몇 시스템 콜은 AIX에서 사용할 수 없는 경우가 있다. 다음은 잘 알려진 사항과 AIX에서 동작할 수 있도록 하는 해결방안에 대해 이야기하고 있다.

Unresolved seteuid(), setegid()

컴파일/링킹시 `setegid()`나 `seteuid()`심볼을 찾을 수 없다는 에러가 발생하면 다음 코드를 추가해야 한다.

```
#include <stdio.h>
#include <unistd.h>
extern "C" {
int seteuid(uid_t);
int setegid(gid_t);
}
```

프로세스 정보 알아내기

Solaris에는 프로세스 정보를 담는 `prpsinfo` 구조체가 있고, `ioctl(fd, request, .../*arg*/)`

함수로 이 구조체를 이용할 수 있다.

```
char path[64];
sprintf(path,"/proc/%lu",(unsigned long)getpid());
int pfd = open(path,O_RDONLY);
struct prpsinfo processinfo;
```

AIX는 Solaris와 동일한 방식으로 프로세스 파일 시스템을 다루지 않으므로, 위와 같은 코드라면 `open()`부분에서 에러가 발생하게 된다. AIX 5.1에는 프로세스 파일 시스템인 `/proc` 이 추가되었지만 세부사항은 Solaris와 다르다. 좀더 자세한 사항은 이 문서의 "프로그래밍 이슈"부분을 참고하시오.

AIX에서 위와 같은 코드는 다음처럼 구현할 수 있다.

```
int pfd = (unsigned long)getpid();
long psize = sysconf(_SC_PAGESIZE);

struct procinfo processinfo;
memset(&processinfo,0,sizeof(processinfo));

while(1){
if(getprocs(&processinfo,sizeof(processinfo),(size_t)0,0,&pfd,0)!= 1){
   .........
}
```

시그널

Solaris에는 특유의 `sig2str()`, `str2sig()` 함수가 있어서 이 함수로 특정 시그널과 해당 시그널 스트링을 서로 변환할 수 있다.

AIX에는 이러한 함수가 없으므로 libc.a (shr.o)에서 반출된 sys_siglist 배열을 참조해야 한다.

```
extern char *sys_siglist[];

int main(int argc, char *argv[])
```

```
{
        char *str;
        int sig;

        sig = atoi(argv[1]);
        str = sys_siglist[sig];
        printf("Signal %d -> \"%s\"\n", sig, str);
}
```

cftime()

AIX는 C 라이브러리에서 `cftime()`함수를 지원하지 않는다.

```
int cftime(char *s, char *format, const timet *clock);
```

그러나 대신 `strftime()`함수가 있으므로 이 함수로 비슷한 기능을 구현할 수 있다.

```
size_t strftime (string,length,*Format,*Tmdate)
char *String;
size_t Length;
const char *Format;
const struct tm *TmDate;
```

`strftime()`서브루틴은 TmDate파라미터에서 가리키는 tm구조체의 시간과 날짜 정보를 가지고 Format 파라미터의 정보에 따라 포맷을 맞추어서 문자열로 변환 후 String 파라미터에서 지정한 포인터로 반환해준다. 포맷정보는 실제로는 LC_TIME 카테고리에서 지정한 현재 세팅에 따른다. tm구조체는 사용자가 지정해 줄 수도 있고 `localtime()`함수나 `gmtime()`함수에서 넘겨준 값일 수도 있다. 결과로 나오는 문자열은 printf Format 파라미터와 비슷하며, String 포인터가 지정하는 메모리에 저장된다. 문자열의 최대 길이는 Length 파라미터에 의해 결정되고 널(NULL) 캐릭터로 끝난다.

dladdr() - 심볼릭 어드레스 계산하기

Solaris의 `dladdr()`은 주소 addr을 포함하는 공유 오브젝트에 대한 정보를 동적 링커(dynamic linker)에게 질의하는 함수이다. AIX에서는 이 함수를 지원하지 않는다.

AIX에서 공유 오브젝트를 open하기 위해 `dlopen()`함수를 사용하며, 현재 수행되고 있는 프로그램의 주소공간에 동적으로 매핑시킨다. 그러나 주소만 가지고 심볼의 이름을 알아낼 수 있는 손쉬운 방법은 제공하지 않는다.

각 함수 뒷부분의 트레이스 백 테이블을 살펴보고(`/usr/include/sys/debug.h` 를 보자) , 명령어 주소를 함수 이름으로 바꿀 수 있는지 잘 읽어봐야 한다.

다음 예제를 보자.

```
$ cat tryit.c
extern int malloc();
main() {
        int *x;
        x = (int *)&malloc;
        printf("Code for malloc is at %p\n", *x);
}
$ cc -o tryit tryit.c
$ ./tryit
```

결과는 다음과 같다.

`Code for malloc is at d016bc64` ---> 이 값을 잘 살펴보자.

다음 이 주소값을 살펴보기 위해 디버거를 사용해야 한다. 간단히 dbx를 사용할 수도 있고, 각 공유 라이브러리의 텍스트 시작위치를 알아내려면 loadquery가 더 좋을 수도 있다.

```
$ dbx tryit
Type 'help' for help.
reading symbolic information ...warning: no source compiled with
-g
(dbx) map
Entry 1:
    Object name: tryit
    Text origin: 0x10000000
```

```
    Text length: 0x67f
    Data origin: 0x200003d8
    Data length: 0xf8
    File descriptor: 0x4

Entry 2:
    Object name: /usr/lib/libcrypt.a
    Member name: shr.o
    Text origin: 0xd00250f8
    Text length: 0x87a
    Data origin: 0xf0870528
    Data length: 0x13c
    File descriptor: 0x5

Entry 3:
    Object name: /usr/lib/libc.a
    Member name: shr.o
    Text origin: 0xd015f720
    Text length: 0x1bed86
    Data origin: 0xf07f1340
    Data length: 0x7ea10
    File descriptor: 0x6
(dbx) q

libc.a(shr.o) starts at 0xd015f720

$ ar -x /usr/lib/libc.a shr.o
$ dump -hv shr.o

shr.o:
               ***Section Header Information***
               Section Header for .text
PHYaddr    VTRaddr    SCTsiz     RAWptr     RELptr
```

```
0x00000000 0x00000000 0x0014fab0 0x00000160 0x001bed9e
.....
.....
```

가상 주소값 VTRaddr 0x00000000 과 RAWptr 값 0x00000160 에 주의하자.

다음 명령어를 사용하여 필요한 값을 얻어보자.

```
$ nm -x shr.o | grep ".malloc" | grep " T "
.__malloc T 0x00008000
.__malloc_init T 0x0000b26c
.__malloc_postfork_unlock T 0x0000b500
.__malloc_prefork_lock T 0x0000b61c
.malloc        T 0x0000c3e4
.malloc_s      T 0x0000d6e0
.malloc_y      T 0x0000ec6c
```

여기에서 .malloc의 nm값은 0x0000c3e4 이다.
이제 여기까지 얻은 모든 정보를 가지고 malloc의 위치를 계산해보자.

memory_address(malloc) = nm(.malloc) + text_origin - virtaddr(.text) + rawoffset(.text)

```
$ bc
ob=16
ib=16
C3E4 + DO015F720 - 0 + 160
D016BC64
```

프로그래밍 이슈

다음은 프로그래밍과 관련된 이슈에 대해 다루고 있다.

IPC 제한

UNIX 프로그래밍에서 여러 프로세스간에 통신하기 위해 IPC 방식(세마포어, 공유 메모리, 메시지 큐)을 사용하게 된다. Solaris에서는 어플리케이션 프로그래머들이 IPC 자원할당을 시스템 관리자에게 요청하고, 시스템 관리자는 이 요청을 받아서 `/etc/system` 파일을 편집하고 IPC 자원(세마포어, 공유 메모리 세그먼트, 메시지 큐)에 대해 제한을 두게 된다. `/etc/system` 파일에서 사용하는 항목은 다음과 같다.

- **shmsys:shminfo_shmmax** - System V 공유메모리 세그먼트의 생성 가능한 최대크기
- **shmsys:shminfo_shmmni** - 시스템 전체에서 생성할 수 있는 공유메모리 세그먼트 개수
- **semsys:seminfo_semmni** - 세마포어 ID의 전체 개수
- **semsys:seminfo_semmns** - System V 세마포어의 최대 개수
- **msgsys:msginfo_msgtql** - 생성할 수 있는 최대 메시지 개수

이러한 방식의 문제점은 개수를 너무 크게 잡으면 커널 싸이즈도 같이 늘어나며, 이 경우 성능이 심각하게 영향 받을 수 있다. 또 개수를 너무 작게 잡으면 실행 시간 중에 문제가 발생할 수 있기 때문에, 이런 경우 실시간이라도 시스템 관리자가 `/etc/system` 의 값을 바꾸고 새 값을 적용하기 위해 시스템을 리부팅해야만 한다.

AIX는 이와 다른 방식을 사용하며, IPC제한을 사용자 수준에서 다룰 수 있다. AIX는 커널 수준에서 IPC자원의 제한을 엄격하게 두고 있으며 각각의 IPC 자원은 정해진 제한 내에서 동적으로 할당/해제된다. 어플리케이션은 항상 커널에서 허가한 최대의 IPC 자원을 얻게 되며, `/etc/system` 파일을 편집하거나 제한수준을 올리기 위해 리부팅할 필요가 없다.

따라서 커널은 IPC자원이 할당된 수준 안에서 사이즈가 커지거나 줄어들며, 성능은 IPC자원의 사용기간에 의해 영향받는다. IPC 제한에 대한 내용을 포함하는 구조체는 `/usr/include/sys/`의 `sem.h`, `msg.h`, `shm.h` 세 개의 파일에 정의되어 있으며, 구조체의 이름은 각각 `seminfo`, `msginfo`, `shminfo`이다.

다음은 AIX 4.2, 4.3.0 4.3.1, 4.3.2 그리고 이후 버전(5.1 포함)에 대한 IPC 자원 제한 테이블이다. 여기 기재된 값들은 모두 변경할 수 없다.

AIX 버전	4.2.0	4.2.1	4.3.0	4.3.1	4.3.2 와 이후 버전
Maximum number of semaphore IDs	4,096	4,096	4,096	4,096	131,072
Maximum semaphores per semaphore ID	65,535	65,535	65,535	65,535	65,535
Maximum operations per	1,024	1,024	1,024	1,024	1,024

semop call					
Maximum undo entries per process	1,024	1,024	1,024	1,024	1,024
Size in bytes of undo structure	8,208	8,208	8,208	8,208	8,208
Semaphore maximum value	32,767	32,767	32,767	32,767	32,767
Adjust on exit maximum value	16,384	16,384	16,384	16,384	16,384

메시지 큐

AIX 버전	4.2.0	4.2.1	4.3.0	4.3.1	4.3.2 와 이후 버전
Maximum message size	65,535	4MB	4MB	4MB	4MB
Maximum bytes on queue	65,535	4MB	4MB	4MB	4MB
Maximum number of message queue IDs	1,024	1,024	1,024	1,024	1,024
Maximum messages per queue ID	1,024	1,024	1,024	1,024	1,024

공유 메모리

AIX 버전	4.2.0	4.2.1	4.3.0	4.3.1	4.3.2 와 이후 버전
Maximum segment size	256MB	256MB	256MB	2GB	2GB
Minimum segment size	1	1	1	11	1
Maximum number of shared memory IDs	4,096	4,096	4,096	4,096	4,096
Maximum messages of segments per process	10	11	11	11	11

공유 메모리 제한과 EXTSHM 사용법

프로세스에서 동시에 사용 가능한 공유 메모리 세그먼트의 개수를 변경하거나 제한을 두면 문제가 발생할 수 있다. 원래 디폴트 값은 11이지만 만약 어플리케이션이 이보다 더 많은 개수를 요구하게 되면 EXTSHM 환경변수를 사용해서 extended shmat capability 기능을 활성화시켜야 한다. 다음은 확장 공유 메모리 기능의 원리에 대해 설명한다.

공유 메모리 영역은 일단 크기가 256M보다 작더라도 디폴트로 256M를 차지한다. 그리고 AIX 4.2.1과 그 이후 버전은 확장 공유 메모리를 허용하는데 이는 좀더 디폴트 사이즈를 줄여서 1byte에서 256M로 자유롭게 공유 메모

리를 잡을 수 있도록 했다. 그러나 주소공간은 그 다음 페이지 영역 바운더리(4096 바이트)까지 차지하게 된다. 확장공유 메모리를 사용함으로써 기본적으로 11개의 공유 메모리 영역의 제한이 없어지게 되었다.

프로세스 환경에서 EXTSHM 환경변수를 ON으로 해놓은 프로세스(EXTSHM=ON)는 확장 공유 메모리 기능을 사용할 수 있다. 이 경우 프로세스는 제한 없이 공유 메모리 영역을 사용할 수 있다. 이전에는 파일매핑이 지원되었으며 여전히 파일매핑으로 256MB(세그먼트 사이즈)의 배수로 주소공간을 차지한다. 이 모드에서는 공유 메모리 영역의 크기를 변경할 수 없으며 커널프로세스도 동일하게 동작한다. 이 환경변수를 ON으로 하지 않으면 11개의 256M 영역만 공유 메모리로 사용할 수 있다.

주의: 256M 보다 더 크기가 큰 공유 메모리 영역을 사용하게 되면, 256M의 배수로 영역을 차지한다. 확장 메모리(shmat) 기능을 사용하더라도 동일하다.

공유 메모리 영역에는 다음과 같은 제한사항이 있다.

- I/O 지원은 메모리 맵 영역과 같은 방식으로 제한된다.
- uphysio() 타입 I/O만 지원된다.(no raw I/O)
- 인터럽트 핸들러에서 버퍼를 해제할 때(unpin) 공유 메모리 영역은 I/O 버퍼로 사용될 수 없다. 예를 들어 이러한 영역은 async I/O 버퍼로 사용할 수 없다.
- plock()서브루틴으로 메모리 맵 세그먼트를 고정(pin)할 수 없기 때문에, plock()서브루틴을 사용해서 세그먼트를 고정할 수 없다.

RPC

AIX는 두 가지의 서로 다른 RPC구현인 OSF RPC와 TIRPC를 지원한다. OSF RPC에 대한 헤더파일은 /usr/include/rpc에 있고, TIRPC에 대한 헤더파일은 /usr/include/tirpc/rpc에 있다.

OSF RPC는 통신하기 위해 BSD 소켓을 사용하고, TIRPC는 TLI 인터페이스를 사용하는데 결국 스트림과 트랜스포트 레이어 인터페이스를 사용하게 된다. TIRPC는 원래 SUN Microsystems에서 개발되었으며 멀티 쓰레드이고 OSF RPC보다 더 많이 사용되고 있다. TIRPC를 사용하려면 어플리케이션을 /usr/lib/libnsl.a 와 링크시켜야 한다.

AIX man 페이지는 OSF RPC에 대한 정보는 제공하지만 TIRPC에 대한 정보는 제공하지 않는다. TIRPC에 대해 좀더 자세히 알아보려면 http://ibm.com/eserver/iseries/developer/tirpc/ 싸이트를 참고하시오.

세마포어와 RealTime 옵션

Solaris는 POSIX와 System V 세마포어 API를 모두 지원하지만, AIX는 System V API 만을 지원한다. AIX가 비록 Unix98 표준을 준수하기는 하지만, Unix98 표준안에서 POSIX realtime 옵션은 단지 옵션일 뿐 반드시 준수해야 하는 사항은 아니다. AIX는 `semaphore.h` 파일에서 `/usr/include` 에 위치하는 파일들도 포함하도록 되어 있는데 이 사항은 IEEE POSIX 1003.1c 에서만 지정하고 있는 사항이다. 따라서 만약 다음 함수를 사용하는 어플리케이션이 있다면 컴파일, 링킹은 제대로 되지만 AIX에서는 실시간에 에러가 발생하게 된다.

```
sem_close()
sem_destroy()
sem_getvalue()
sem_init()
sem_open()
sem_post()
sem_trywait()
sem_unlink()
sem_wait()
```

위 API는 지원되지 않으며 호출하면 –1이나 errno=ENOSYS 를 반환한다.

예제

다음은 `sem_init()`를 사용한 예제이다.

```
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <errno.h>
main ()
{
        sem_t mysem;
        int pshared = 0;
        int rc;
        char errmsg[256];
```

```
        rc = sem_init (&mysem, pshared, 0);
        printf("Errno = %d\n", errno);
        if (rc)
                printf ("sem_init failed %ld: '%s'\n", rc, strerror(errno));
}
```

이 프로그램을 libc.a 와 링크하면 실행시간에 다음과 같은 에러가 발생한다.

```
Errno = 109
sem_init failed -1: 'Function not implemented'
```

Errno 109는 ENOSYS이며 이 값은 `errno.h` 파일에 정의되어 있다.

```
#define ENOSYS 109 Function not implemented POSIX
```

세마포어 해법

만약 어플리케이션에서 위에서 언급한 지원되는 않는 API를 사용하고 있다면 UNIX98 호환을 보장하기 위해 다음 몇 가지 옵션을 고려해볼 수 있다.

1. System V 에 해당하는 `msem_init()`, `msem_lock()`, `msem_unlock()`, `msem_remove()` 등의 함수를 사용한다. 이 함수는 헤더파일 `<sys/mman.h>` 에 선언되어 있다.
2. `pthread_mutex_init()`, `pthread_mutex_lock()`, `pthread_mutex_unlock()`, `pthread_mutex_destroy()` 함수를 사용한다.
3. 네이티브 세마포어 구현을 사용한다. 자기 자신의 바이너리 세마포어 함수를 정의하기 위해 `semget()`, `semctl()`, `semop()` 를 사용한다. 이 함수들은 헤더파일 `<sys/sem.h>` 에 선언되어 있다.

Solaris는 이외에도 자기 자신만의 API인 `sema_destroy()`, `sema_init()`, `sema_post()`, `sema_wait()`, `sema_trywait()` 를 지원하며 이 함수는 `libthread.a` 라이브러리를 링크해야 하고 Solaris 쓰레드와 함께 사용한다. 위에서 언급한 해결옵션은 AIX에서 지원하지 않는 Solaris 세마포어 API에 대한 해결방법으로 사용할 수 있다.

스택 프레임 이슈

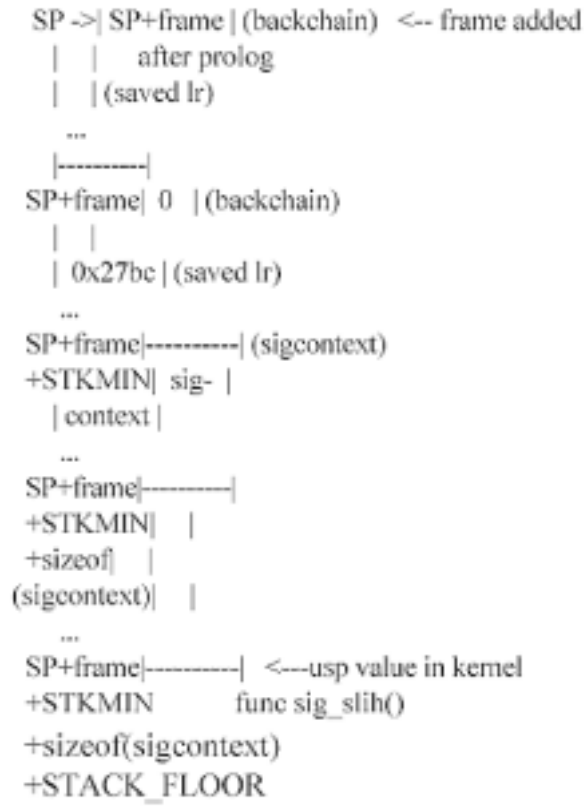
Solaris API는 스택 프레임을 지원한다. 다음은 예제이다.

```
ostr << "Signal " << signo << " encountered. Stack dump follows:" << endl;
```

```
ostr << "(This stack dump is not precise - look for a core file.)" << endl;
mcontext_t *thiscontext = &uap.uc_mcontext;
// For each function on the stack, print the "where" info for it.
        struct frame *f = (struct frame *)thiscontext->gregs[REG_SP];
        for (int i=1; f && f->fr_savpc; f = f->fr_savfp ){
                ostr << "\t[" << i++ << "] ";
                Dl_info dlinfo;
                if (dladdr((void *)f->fr_savpc,&dlinfo) != 0){
                        if(cplus_demangle((char *)dlinfo.dli_sname, buf,
                sizeof(buf)) == DEMANGLE_ESPACE)
                        ostr << (char *)dlinfo.dli_sname;
                else
                        ostr << (char *)buf;
```

다음은 AIX에서 스택 프레임을 구현하기 위한 해법이다. 우선 시그널 핸들러안의 어플리케이션 <stack> 레이아웃을 보자. 어플리케이션이나 쓰레드가 시그널을 받았을 때 컨텍스트 정보는 user <stack>에 저장된다. 시그널 컨텍스트는 SP+STKMIN(sys/pseg.h 참고)위치에 저장된다. 시그널 컨텍스트에서 <stack>포인터, iar, lr, 그외 값(sys/context.h 참고)을 얻을 수 있다. 우선 <stack>포인터 값을 알아내면 나머지 값은 주소값을 계산하여 차례차례 알아낼 수 있다.

아래의 스택 프레임 다이어그램을 보고 어플리케이션이 시그널 핸들러에 있을 때 stack trace를 알아낼 수 있다.



pseg.h와 signal.h 헤더 파일을 참고할 수 있다.

```
#include <stdio.h>
#include <ucontext.h>
#include <signal.h>
typedef struct stack_frame {
    struct stack_frame *next;
    int unused;
    int pc;
} stackFrame;
void stackTrace()
{
    stackFrame *sf;
    (void)getcontext(&uc);
    for(sf=(stackFrame*)uc.uc_mcontext.jmp_context.gpr[1]; sf !=NULL; sf = sf-
```

```
    >next)
    {
        fprintf(stdout, "sf=%p, pc=%p, unused=%p\n", sf, sf->pc, sf->unused);
    }
}
```

/proc 파일 구조

Linux의 /proc, Solaris의 /proc, AIX의 /proc 파일 시스템 구조는 모두 다르다.(정해진 표준이 없기 때문에) AIX의 /proc 디렉토리 구조는 Solaris의 /proc과 다른데, AIX는 Solaris 디렉토리 구조를 지원하지 않는다. (예를 들어 AIX에는 /proc/self/* 디렉토리가 없다) 만약 /proc 디렉토리를 사용하려고 하면 해당 프로세스의 pid로 /proc/<pid>디렉토리를 이용할 수 있다. /proc/<pid>/status와 /proc/<pid>/psinfo 파일에서 각각 프로세스 usage/info를 얻어올 수 있다. 이러한 어플리케이션을 작성할 때는 AIX에서는 sys/procfs.h 헤더파일을 포함해야 한다.

Solaris에서 어플리케이션은 /proc/pid/ctl 파일을 사용하여 micro-state 어카운팅 정보를 on/off 할 수 있다. 따라서 PR_MSACCT 플랙이나 이와 유사한 다른 변수가 필요 없다. AIX는 /proc의 어카운팅 정보를 얻기 위해 /proc/<pid>/status파일에서 pr_utime, pr_stime, pr_cutime, pr_cstime 필드를 유지한다. 이 값은 초 단위의 정밀성을 가지며 최소단위는 nano-second이다.

/proc 및 디렉토리 구조에 대한 좀더 자세한 정보는 AIX 5L Version 5.1 Files Reference 를 참고하시오.

메모리 매핑 파일

메모리 매핑은 파일을 메모리로 매핑시키며 좀더 빠른 I/O를 구현하는데 사용된다. 파일을 매핑시키는 함수는 mmap이며 함수 정의는 다음과 같다.

```
mmap( baseAddr, length, protection, flags, fileDescriptor, offSet )
```

baseAddr – 파일을 매핑할 주소값

length – 매핑할 바이트 수. OS는 가장 근접한 숫자의 페이지를 할당하게 된다.

protection – 매핑된 영역의 접근 퍼미션

flags – 매핑된 영역의 특성

fileDescriptor – 메모리에 매핑하기 위해 open한 파일의 디스크립터

offSet – 파일이 매핑될 영역의 baseAddr부터의 옵셋

Solaris에도 mmap은 있지만 AIX와 약간 다르게 동작한다. Flags 아규먼트에는 MAP_FIXED나 MAP_VARIABLE같은 값을 지정할 수 있다. MAP_FIXED로 지정하면 가능한 매핑가능한 영역을 baseAddr로 맞추게 되고 가능하지 않은 경우 에러가 발생한다. MAP_VARIABLE로 지정하면 OS는 가능한 baseAddr로 매핑하려고 하지만 가능하지 않은 경우 매핑할 수 있는 다른 주소값을 반환하게 된다.

첫번째 문제점은 Solaris와 AIX가 서로 다르게 동작한다는 점이다. 프로세스가 MAP_FIXED로 파일을 특정 메모리 영역 0x80000000에 매핑하는 경우를 가정해보자. 매핑한 다음 동일한 파일 혹은 다른 파일을 같은 위치 0x80000000에 매핑하려 한다고 해보자. Solaris는 에러가 발생하지 않은 채로 함수가 반환되고 AIX는 다음과 같은 에러를 내며 실패하게 된다.

ENOMEM – 지정한 길이만큼의 바이트를 매핑할 만한 충분한 주소 공간이 없거나, 어플리케이션이 X/Open UNIX95 호환요청을 하지 않은 경우이다. 혹은 MAP_FIXED 플랙을 지정한 다음 지정한 주소공간(baseAddr, baseAddr + length)의 일부분이 이미 할당되어 있는 경우이다.

물론 파일을 매핑한 다음 같은 메모리 위치에 파일을 해제하지 않은 채로 다시 매핑하는 것은 문제를 일으킬 여지가 있으므로 어플리케이션에서 이러한 상황을 잘 다루어야 한다.

두 번째 문제점은 메모리에 파일을 어떻게 매핑시킬 것인가 하는 레이아웃 문제이다. 두개의 파일을 메모리에 매핑시키려는 어플리케이션을 가정해보자. 첫번째 파일은 MAP_FIXED로 0x80000000에 매핑했고, 두 번째 파일은 주소 0x0을 사용하고 있다. 0x0에 파일을 매핑하려고 하면 OS는 적당한 주소공간을 골라서 거기에 매핑하게 된다. 이런 경우 두 번째 파일은 첫번째 파일이 매핑된 바로 다음에 매핑되게 된다. 그러나 나중에 어플리케이션이 첫번째 파일이 차지하는 매핑 영역을 늘리려고 하는 상황을 가정해보자. 어플리케이션이 첫번째 파일을 내리고 새로운 더 큰 파일을 0x80000000에 매핑하려고 한다. 이런 경우 mmap은 실패한다. 이 경우 원인은 두 번째 파일과 두 번째 파일이 매핑된 위치에 있다. 두 번째 파일은 0x0위치로 매핑하기로 했기 때문에, AIX는 0x80000000에 매핑된 첫번째 파일의 뒤에 두 번째 파일을 매핑하기로 결정한다. 나중에 첫번째 파일보다 더 큰 파일을 새로 0x80000000에 매핑하려고 하면 기존의 남겨진 영역이 충분하지 않기 때문에 에러가 발생한다.

세 번째 문제점은 잘 이해가 안되겠지만 파일을 언매핑(unmap)하는 문제와 관련되어 있다. AIX에서는 원하는 만큼 영역을 언매핑하고 그 영역을 OS에 반환한 다음 다시 에러 없이 영역을 매핑할 수 있다. 위에서 언급한 두개의 파일을 연속해서 매핑하는 경우를 생각해보자. 어플리케이션은 두개의 파일이 연속되어 있기 때문에 0x80000000에서부터 시작해서 두 번째 파일이 매핑된 영역 끝까지 한번에 munmap할 수 있다. 별로 권장할만한 방법은 아니지만 어쨌든 AIX에서는 이러한 방식을 허용하고 있다. AIX는 매핑한 원래 파일의 크기를 따로 저장하지 않기 때문에 언

매핑할 때 이를 고려하지 않는다. 이 경우 매핑한 영역보다 더 적은 영역을 언매핑하게 되면 문제가 발생할 수 있다. munmap은 이전에 언매핑(unmap)한 크기가 매핑한 크기보다 작거나 이와 동일한 경우에 성공하게 된다. 동일한 주소에 더 큰 크기의 파일을 매핑하게 되면, 일부분이 이미 언매핑되어 있는 상태가 아니기 때문에 에러가 발생하게 된다.

자바 이슈

Solaris에서 AIX로 자바 관련 어플리케이션을 포팅하기 위해서 Porting Java Applications to AIX 문서를 참고하시오.

자바 어플리케이션에서는 1:1 쓰레드 모델만을 지원하며 AIXTHREAD_SCOPE=S 환경변수를 설정해야 한다. 추가로 성능을 높이거나 제대로 동작하기 위해서 다음 환경변수를 추가로 설정해야 한다.

```
AIXTHREAD_RWLOCK_DEBUG=OFF
AIXTHREAD_MUTEX_DEBUG=OFF
AIXTHREAD_COND_DEBUG=OFF
```

쓰레드 모델 이슈

AIX에서 기본 쓰레드 모델은 M:N 이며, M개의 사용자 쓰레드가 N개의 LWP(Light Weight Process)나 커널 쓰레드로 매핑된다. 이 모델은 가장 복잡하지만 성능이 좋은 모델이다. 그러나 상황에 따라 이 모델이 어플리케이션에 가장 효과적인 모델이 아닐 수도 있다. 만약 쓰레드 어플리케이션이 실행시간에 의도와 다른 동작을 하게 된다면 다음 환경변수를 설정해보자.

```
AIXTHREAD_SCOPE=S
```

이 환경변수는 쓰레드 모델을 1:1로 바꾸며 한 개의 사용자 쓰레드는 한 개의 커널 쓰레드로 매핑된다.

성능

쓰레드 어플리케이션의 성능을 높이기 위해 다음 환경 변수들을 바꿔보자.

```
MALLOCMULTIHEAP=1
SPINLOOPTIME=500
YIELDLOOPTIME=500
AIXTHREAD_SLPRATIO=1:1
AIXTHREAD_RWLOCK_DEBUG=OFF
AIXTHREAD_MUTEX_DEBUG=OFF
AIXTHREAD_COND_DEBUG=OFF
```

malloc서브 시스템을 많이 사용하는 멀티 쓰레드 사용자 프로세스의 경우 어플리케이션을 시작하기 전에 `MALLOCMULTIHEAP=1` 을 설정하면 성능이 좋아진다. 특히 멀티쓰레드 C++ 어플리케이션의 경우 constructor/destructor를 많이 사용하며 이때 malloc을 사용하기 때문에 성능이 많이 좋아진다. 멀티 쓰레드 사용자 프로세스를 SMP머신에서 수행할 때 성능이 더 좋아지며 시스템 스코프 쓰레드를 사용할 때 더더욱 해당된다. 그러나 어떤 경우 다른 상황에서 성능이 좋아질 수 있으며 프로세스가 한 개인 경우 이러한 가능성이 높아진다.

`SPINLOOPTIME`은 프로세스를 반납하기 위해 커널을 호출하는 등의 두 번째 동작을 취하지 않은 채로, busy mutex나 spin lock을 얻기 위해 시스템이 몇 번 시도를 할 것인지를 결정한다. 이 변수는 MP머신에서 효과가 있으며, 활동하는 pthread가 자기가 들고 있는 락을 해제할 때 발생한다.

`YIELDLOOPTIME`은 시스템이 락을 가지고 sleep 상태로 들어가기 전 busy mutex나 spin lock을 얻으려고 할 때 프로세서를 반납하는 횟수를 결정한다. 수행되고 있는 다른 쓰레드가 더 높은 우선순위를 가지고 있을 때 프로세서는 다른 커널 쓰레드로 반납된다. 이 변수는 여러 개의 락을 사용하는 복잡한 어플리케이션에서 효과적이다.

`AIXTHREAD_SLPRATIO`는 sleep 상태로 들어간 pthread를 지원하기 위해 유지해야 하는 커널 쓰레드의 비율이다. 일반적으로 sleep 상태로 들어간 pthread는 한번에 하나씩 만 깨어나기 때문에 이를 유지하는 커널 쓰레드는 아주 적은 개수만으로 충분하다.

pthread 라이브러리는 현재 활동중인 mutex, condition 변수, 디버거에서 사용하는 read/write 락에 대한 모든 목록을 관리한다. mutex, condition 변수, 디버거에서 사용하는 read/write락을 생성하게 되면 이와 관련되어 어느 정도의 오버헤드가 생긴다. `AIXTHREAD_RWLOCK_DEBUG, AIXTHREAD_MUTEX_DEBUG, AIXTHREAD_COND_DEBUG` 변수를 `OFF`로 설정하면 앞서 언급한 오버헤드가 없어지게 되고, mutex, condition 변수, read/write락을 많이 사용하는 어플리케이션의 경우 성능이 더 좋아진다.

지원여부

AIX용 fix와 update는 다음 싸이트에서 다운로드받을 수 있다.
http://techsupport.services.ibm.com/rs6000/fixes

결론

더 자세한 정보는 참고문헌의 문서를 참조하시오.

부록 Solaris 와 AIX 명령어 비교

부록에서는 AIX 4.3.3과 Solaris8 의 시스템 관리상 필요한 명령어들을 비교하여 수록했다. 기본적인 시스템 관리 명령은 AIX 4.3.3이나 AIX 5.1, AIX 5.2 모두 비슷하므로 버전업에 따른 심각한 차이점은 거의 없다.

명령어들은 사용방법에 따라 주요 카테고리로 나누어 수록했으며 연관된 중요 파일 및 정보도 수록되어 있다. AIX OS에 대한 좀더 자세한 정보는 http://www-1.ibm.com/servers/aix/library/ 의 Technical Publications에서 찾아볼 수 있다.

패키징

단위	AIX 4.3.3	Solaris 8
최소 설치 단위	fileset	package
단독으로 설치 가능한 이미지 - 한 개의 설치단위로 배포할 수 있는 단위	package	package
패키지를 묶어서 만든 단위 그룹	bundle	software cluster
패키지를 묶어서 만든 단위 그룹과 소프트웨어 클러스터	제공되는 bundle. 예) App-Dev : 어플리케이션 개발환경 관련 Client : ▸ Pers-Prod ▸ DCE-Client ▸ Media-Defined	Software configuration clusters 예) ▸ Core : 기본 OS에 필요한 파일 ▸ End-User System Support: Core와 windows 환경 ▸ Developer System Support: End-User System Support와 개발환경 ▸ Entire Distribution: Developer System Support에 기능추가 ▸ Entire Distribution Plus OEM: Entire Distribution에 third party 하드웨어 드라이버 추가 (SPARC에만 해당)

설치와 업그레이드

기능	AIX 4.3.3	Solaris 8

패키지 설치	`installp -a`
	smitty를 사용할 경우:
	`smitty install_latest`
		`pkgadd`

설치된 패키지 보기	`lslpp -L`
	smitty를 사용할 경우:
	`smitty list_installed_sw`
		`pkginfo`
		또는
		`pkgparam`

설치된 패키지 삭제하기	`installp -r`
	smitty를 사용할 경우:
	`smitty reject`
	
	`installp -u`
	smitty를 사용할 경우:
	`smitty remove`
		`pkgrm`

설치가 제대로 되었는지 확인하기	`lppchk`
	또는 fast path:
	`smitty check_files`
		`pkgchk`

패치 설치하기	`instfix`
	또는 fast path:
	`smitty update_by_fix`
		`patchadd`

패치 삭제하기	`installp -r`
	또는 fast path:
	`smitty reject`
		`patchrm`

설치된 패치 확인하기	`instfix -ia`	`showrev -p`
다른 디스크에 OS 설치하기	`alt_disk_install`	Live Upgrade
네트워크 설치를 할 수 있도록 설치용 서버 구성하기	`nimconfig`	`setup_install_server` *`install_dir_path`*
네트워크 설치를 할 수 있도록 부트 서버 구성하기	`smitty nim_config_env`	`setup_install_server -b` *`bootdirpath`*
네트워크 설치용 클라이언트 구성하기	`nim -o bos_inst`	`add_install_client`

부팅/셧다운

기능/위치	AIX Version 4.3.3	Solaris 8
부팅 과정	Read Only Storage (ROS) : 시스템의 마더보드를 검사한 다음 POST(Power-On Self-Test)를 수행한다. 다음 부트 이미지의 위치를 찾은 다음 부트 이미지를 메모리에 올린다. 시스템 초기화 작업을 시작한 다음 /etc/rc.boot의 1단계를 수행한다. 기본 디바이스 환경설정 : 기본 디바이스들의 환경을 설정하기 위해 환경설정 매니저를 수행한다. 시스템 부트 : init 프로세스의 2번째 단계를 시작한다. 다음 하드디스크의 루트 파일 시스템을 마운트한 다음, /etc/inittab 파일에 정의되어 있는 프로세스를 수행한다. 다음 /etc/rc.boot의 3단계를 수행한다.	PROM 부트 : 시스템 정보를 보여준 다음, POST를 실행한다. bootblk를 메모리에 로드한 다음 ufsboot의 위치를 찾는다. 부트 프로그램 : bootblk를 로드하고 ufsboot를 수행한다. 커널 초기화 : ufsboot를 로드하고 코어 커널을 수행한다. 다음 코어 커널 데이터를 초기화하고 /etc/system 파일에 기재된 대로 다른 커널 모듈을 로드한다. 다음 /sbin/init 프로그램을 시작한다. 초기화 : /etc/inittab 에 기재된 여러 프로세스들을 수행한다.

커널 모듈 디렉토리	커널과 커널 익스텐션 모듈은 다음 디렉토리에 저장된다.
	`/usr/lib/boot`
	`/usr/lib/drivers`
		커널 모듈은 다음 디렉토리에 저장된다. `/platform/sparc/kernel` 이나
		`/platform/i86pc/kernel`
		`/kernel`
		`/usr/kernel`

/etc/inittab 파일에 따라 현재 시스템 런레벨에 맞추어 프로세스, 서비스를 시작하거나 중단하기	/etc/rc 파일에 정의된 대로 디폴트 환경 변수의 값을 정해준다.	/etc/default/init 파일에 정의된 대로 디폴트 환경 변수의 값을 정해준다.
시스템 런레벨	정의되어 있는 런레벨: 0-1 : 나중을 대비하여 예약해 놓음 2 : NFS자원을 공유상태로 사용할 수 있는 다중 사용자 상태(디폴트 런레	8개 런레벨: 0 : Power-down 상태 s/S : 단일 사용자 상태 1 : 관리모드 상태

	벨) 3 : 사용자의 선호도에 따라 정의해 놓음 m,M,s,S : 단일 사용자 상태(유지보수 상태) a,b,c : 현재 수행되고 있는 프로세스는 현재 런레벨로 둔 채 새 런레벨로 프로세스를 시작한다. q,Q : /etc/inittab 파일을 새로 검사하여 시작한다. 주의: 1에서 9 런레벨을 지정했을 때, init 명령어는 현재 레벨의 모든 프로세스를 종료하고, /etc/inittab 파일에 지정된 새 런레벨에 맞추어 프로세스를 시작한다.	2 : 다중 사용자 상태 3 : NFS자원을 공유상태로 사용할 수 있는 다중 사용자 상태(디폴트 런레벨) 4 : Alternative multiuser(사용되지 않음) 5 : Power-down 상태 6 : 리부트 상태
시스템 런레벨 알아보기	`who -r`	`who -r`

시스템 런레벨 바꾸기	`telinit` *`level number`*	다음 중 한가지 방법을 선택할 수 있다.
		`halt`
		`init`
		`poweroff`
		`reboot`
		`shutdown`
		`telinit`
		`uadmin`

스크립트 시작하기	`/etc/rc`	`/sbin/rc` *`run-level number`*
부트 정보 보여주기	`bootinfo`	해당사항 없음
부트 디바이스 순서를 변경하거나 보여주기	`bootlist`	`boot`

사용자 관리

기능/파일 위치	AIX Version 4.3.3	Solaris 8
GUI환경에서 여러 작업을 수행하려면	`smit` (GUI가 아닌 환경에서는	`admintool`

	smitty) 또는 wsm	
사용자 추가	mkuser	useradd
사용자 삭제	rmuser	userdel
사용자 변경	chuser	usermod
사용자 보여주기	lsuser	listusers
패스워드 파일	/etc/passwd 과 /etc/security/passwd	/etc/passwd 과 /etc/shadow
그룹 파일	/etc/group 과 /etc/security/group	/etc/group
사용자당 프로세스 자원 제한하기	/etc/securty/limits	/etc/system
시스템 전체 환경파일	/etc/environment	해당사항 없음
사용자당 환경 특성	/etc/security/environ	해당사항 없음
시스템 로그인시 환경설정 정보	/etc/security/login.cfg	/etc/default/login
사용자 인증 관련된 환경설정 정보	/etc/security/login.cfg	/etc/pam.conf
프로파일 템플리트	/etc/security/.profile	/etc/skel/local.profile

디바이스 관리

기능	AIX Version 4.3.3	Solaris 8
GUI환경에서 여러 작업을 수행하려면	smit (GUI가 아닌 환경에서는 smitty) 또는 wsm	admintool
디바이스 환경설정	cfgmgr	다음 중 한가지 방법을 선택할 수 있다. drvconfig devlinks disks tapes ports
디바이스 정의하기	mkdev	다음 중 한가지 방법을 선택할 수 있다. drvconfig devlinks disks tapes

		ports
디바이스 삭제하기	rmdev	rem_drv
디바이스 변경하기	chdev	해당사항 없음
디바이스 나열하기	lsdev	sysdef
디바이스 보여주기	lscfg	prtconf

네트워크 관리

기능	AIX Version 4.3.3	Solaris 8
GUI환경에서 여러 작업을 수행하려면	smit (GUI가 아닌 환경에서는 smitty) 또는 wsm	admintool
TCP/IP 환경설정하기	mktcpip	ifconfig 또는 vi /etc/nsswitch.conf (직접 편집)
인터페이스 세팅 보여주기	ifconfig	ifconfig
인터페이스 환경설정하기	ifconfig	ifconfig
네임 서비스 변경하기	chnamsv	vi /etc/nsswitch.conf
네임 서비스 환경설정 해제하기	rmnamsv	vi /etc/nsswitch.conf
네임 서비스 보여주기	lsnamsv	cat /etc/nsswitch.conf

프린터 관리

기능	AIX Version 4.3.3	Solaris 8
GUI환경에서 여러 작업을 수행하려면	smit (GUI가 아닌 환경에서는 smitty) 또는 wsm	admintool
프린터 추가하기	mkdev	lpadmin
프린터 시작하기	큐 시작하기 : qadm	enable
프린터 중지하기	큐 중지하기 : qadm	disable
프린터 클래스 추가하기	해당사항 없음	lpadmin
프린터 큐 상태 보기	lpstat	lpstat
프린터 작업 취소하기	qcan	cancel
프린터 큐 추가하기	다음 중 한가지 방법을 선택할 수 있다. mkque mkquedev	lpadmin

	mkvirprt	
프린터 큐 변경하기	다음 중 한가지 방법을 선택할 수 있다. chque chquedev chvirprt	lpadmin
프린터 큐 삭제하기	다음 중 한가지 방법을 선택할 수 있다. rmque rmquedev rmvirprt	lpadmin
프린터 큐 세팅 보여주기	다음 중 한가지 방법을 선택할 수 있다. lsque lsquedev lsvirprt	lpadmin

파일시스템 관리

기능	AIX Version 4.3.3	Solaris 8
GUI환경에서 여러 작업을 수행하려면	smit (GUI가 아닌 환경에서는 smitty) 또는 wsm	해당 사항 없음
디스크 포맷하기	해당 사항 없음 - 자동 포맷 되어 있음	format
파일 시스템 검사하기	fsck	fsck
파일 시스템 마운트 하기	mount	mount
사용 가능한 파일 시스템 공간 보여주기	df	df
디스크 파티션 하기	해당 사항 없음 - 자동 파티션되어 있음	format
디스크의 콘텐츠 테이블 보여주기	lchangelv	prtvtoc
파일 시스템 생성하기	crfs	newfs 혹은 mkfs
파일 시스템 내리기	umount	umount
파일 시스템/파일/디렉토리 백업하기	backup	ufsdump
파일 시스템/파일/디렉토리 풀기	restore	ufsrestore
파일 시스템 변경하기	chfs	tunefs
파일 시스템 삭제하기	rmfs	해당 사항 없음

파일 시스템 보여주기	`lsfs`	`cat /etc/vfstab`

디스크 관리

기능	AIX Version 4.3.3	Solaris 8
GUI환경에서 여러 작업을 수행하려면	`smitty chjfs` 또는 `wsm`	`metatool`

파일 시스템 확장하기	`chfs`
	또는
	`smitty chjfs`
		`growfs`

메타 디바이스 삭제하기	해당 사항 없음	`metaclear`
메타 디바이스 환경설정하기	해당 사항 없음	`metainit`
메타 디바이스 수정하기	해당 사항 없음	`metaparam`
메타 디바이스 이름 바꾸기	해당 사항 없음	`metarename`
메타 디바이스 상태 보기	해당 사항 없음	`metastat`

logical volume 관리

다음 테이블에서 Solaris8은 Veritas Volume Manager (VxVM)을 사용하며, AIX 4.3.3은 AIX자체의 Logical Volume Manager (LVM)를 사용하고 있다.

기능	AIX Version 4.3.3	Solaris 8
스토리지 구조	디스크는 PP(physical partition)으로 구성된다. PV(physical volume)은 디스크와 동일하다. VG(volume group)은 PV(physical volume)로 구성된다. VG(volume group)은 LV(logical volume)으로 나누어 진다. FS(file system)은 LV(logical volume) 상에 위치한다.	디스크는 파티션(partition/slice)으로 구성된다. FS(file system)은 파티션(partition)상에 놓인다. 서브디스크(subdisk : AIX의 PP(physical partition)와 비슷한 개념)는 파티션(partition/slice)으로 구성된다. 플렉스(plex: AIX의 LP(Logical Partition)과 비슷한 개념)는 서브디스크 (subdisk)로 구성된다. 볼륨(volume : AIX 의 LV(logical volume)과 비슷한 개념)은 플렉스(plex)로 구성된다.

		VM디스크는 서브디스크(subdisk)로 구성된다. 디스크 그룹(disk group : AIX의 VG(volume group)과 비슷한 개념)은 VM disk로 구성된다.
GUI환경에서 여러 작업을 수행하려면	`smit` (GUI가 아닌 환경에서는 `smitty`) 또는 `wsm`	`vxva`
LV(Logical Volume)를 다른 PV(Physical Volume)로 이동	`migratepv`	`vxassist move`
LV(Logical Volume) 생성하기	`mklv`	`vxassist make`
LV(Logical Volume) 늘리기	`extendlv`	`vxassist growto`
LV(Logical Volume) 삭제하기	`rmlv`	`vxedit rm`
VM disk상에 sysboot 정보를 생성하기	해당 사항 없음	`vxbootsetup`
VM disk 관리하기	해당 사항 없음	`vxdisk`
볼륨그룹 생성하기	`mkvg`	`vxdg init`
볼륨그룹에서 디스크 제거하기	`reducevg`	`vxdg rmdisk`
볼륨 관리자 상에서 디스크 추가하기	`extendvg`	`vxdiskadd`

디스크 관리하기	`reducevg`
	또는
	`extendvg`
		`vxdiskadm`

디스크 셋업하기	`extendvg`	`vxdisksetup`
LV(Logical Volume) 세팅 변경하기	`chlv`	`vxedit set`

저장장치 구조용 환경설정 레코드 생성하기	`mkvg`
	또는
	`mklv`
		`vxmake`

플렉스(plex)나 볼륨 그룹 관리하기	`chvg`
	또는
	`mkvg`
		`vxplex`

볼륨 그룹 보여주기	`lsvg`	`vxprint`
볼륨 크기 변경하기	`lchangelv`	`vxresize`
PV(Physical Volume)나	`chpv`	`vxsd`

서브디스크(subdisk) 관리하기		

저장장치 구조 상태 보여주기	다음 중 한가지 방법을 선택할 수 있다.
	`lspv`
	`lsvg`
	`lslv`
		`vxstat`

볼륨 관리하기	다음 중 한가지 방법을 선택할 수 있다.
	`chlv`
	`mklv`
	`rmlv`
		`vxvol`

OS 백업하기	`mksysb` (to tape or file) 또는 `mkcd` (CD-ROM)	Solstice Backup: `nwadmin`

OS 백업 풀기	`mksysb` (to tape or file)
	또는
	`mkcd` (CD-ROM)
		다음 Solstice Backup 중 선택할 수 있다 :
		`nwadmin`
		`nwrecover`

Troubleshooting

기능/파일 위치	AIX Version 4.3.3	Solaris 8

호스트 이름 변경하기	`chdev -l inet0 -a hostname=`*`host name`*	다음 파일들을 변경한다.
		`/etc/nodename`
		`/etc/hosts`
		`/etc/hostname.*`
		`/etc/net/*/hosts`

알려진 네트워크 서비스와 해당 포트번호 목록	`/etc/services`	`/etc/services`
프로토콜 목록	`/etc/protocols`	`/etc/protocols`
인터넷 프로토콜용	`iptrace`	`snoop`
네트워크 상태 보기	`netstat`	`netstat`
NFS와 RPC 상태 보기	`nfsstat`	`nfsstat`
네트워크 I/O와 네트워크 CPU 사용량 상태 보기	`netpmon`	해당 사항 없음
가상 메모리의 스냅샷 보기	`svmon`	해당 사항 없음

가상 메모리 사용량 보기	vmstat	vmstat
I/O 사용량 보기	iostat 또는 filemon	iostat
시스템 상황에 대한 리포트 보기	sar	sar
시스템 내에서의 lock상황보기	lockstat	해당 사항 없음
CPU 사용량 보기	tprof	해당 사항 없음
성능시험을 위해 시스템이 다른 용량의 메모리를 사용하는 것으로 시험해보기	rmss	해당 사항 없음
시스템 에러 로그 보기	errpt -a	dmesg
덤프 디바이스 보기/지정하기	sysdumpdev	해당 사항 없음
페이징/스와핑 스페이스 보기	lsps -a	swap -l
cron에 접속할 수 있는 사용자 지정하기(만약 파일이 아예 없다면 모든 사용자가 cron에 접속할 수 있다)	/var/adm/cron/cron.allow	/etc/cron.d/cron.allow
cron에 접속할 수 없는 사용자 지정하기	/var/adm/cron/cron.deny	/etc/cron.d/cron.deny
로칼 시스템에서 명령어를 수행할 수 있는 원격 시스템이나 사용자 지정하기	/etc/hosts.equiv	/etc/hosts.equiv
원격 파일 시스템 혹은 로컬 파일 시스템을 마운트 하기 위한 정보	/etc/filesystems	/etc/vfstab
현재 마운트되어 있는 파일 시스템의 정보 보기	mount	mount 또는 pg /etc/mnttab
디폴트 수퍼유저 로그	/var/adm/sulog	/var/adm/sulog
syslogd 로긴 환경설정하기	/etc/syslog.conf	/etc/syslog.conf
물리적 RAM 양 보기	bootinfo -r	prtconf

참고문서

- Migrating Solaris Applications to AIX
http://www-1.ibm.com/servers/esdd/articles/solaris_aix.html

- AIX 5L Porting Guide Redbook
ibm.com/redbooks — “SG24-6034-00"로 검색할 것

- AIX Linking and Loading Mechanisms paper
http://ibm.com/servers/esdd/pdfs/aix_ll.pdf

- AIX 온라인 다큐먼트 — "Program Address Space Overview" 혹은 "Understanding Memory Mapping"로 검색할 것
http://publib.boulder.ibm.com/cgi-bin/ds_form

- Performance Management Guide online documentation
http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixbman/prftungd/2365c15.htm

- Information on TIRPC
http://ibm.com/eserver/iseries/developer/tirpc/

- AIX 5L Version 5.1 Files Reference book (/proc 디렉토리와 디렉토리 구조에 대한 자세한 정보를 알 수 있음)
http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/files/aixfiles/proc.htm

- Porting Java Applications to AIX paper
http://ibm.com/servers/esdd/articles/porting_java/index.html

- C and C++ Application Development on AIX
ibm.com/redbooks — “SG24-5674-00"로 검색할 것

- 공유라이브러리 링크하는 방법, 디바이스 드라이버 작성하는 방법에 대한 예제코드
http://www.developer.ibm.com/tech/samples.html

- AIX Version 4.3 Migration Guide
ibm.com/redbooks — “SG24-5116-00"로 검색할 것

- AIX 5L Base Documentation
http://publibn.boulder.ibm.com/cgi-bin/ds_form

- Quick reference: Solaris to AIX
ibm.com/redbooks — “REDP0104"로 검색할 것

- Kernel extensions and device driver support
http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixprggd/kernextc/kernextctfrm.htm

- Solaris Internals
By Jim Mauro and Richard McDougall
ISBN 0-13-022496-0

TSS DIY 출간 내역

▶ AIX DIY

AIX DIY 1st Edition	(As of 1999. 6)
AIX DIY 2nd Edition	(As of 1999. 11)
AIX DIY 3rd Edition	(As of 2000. 6)
AIX DIY 4th Edition	(As of 2001. 1)
AIX5L DIY Special Edition	(As of 2001. 5)
AIX5L V5.2 Update	(As of 2002. 11)

▶ AIX DIY Special Edition

컴퓨터 아키텍처 정리	(As of 2000. 6)
클러스터 컴퓨팅	(As of 2000. 12)
RS/6000 초보자를 위한 TechKit (기술지원 가이드)	(As of 2001. 5)
AIX PerfKit (Performance Tuning Guide)	(As of 2001. 6)
HP/UX to AIX Porting Guide	(As of 2001. 10)
SUN/Solaris to AIX Porting Guide	(As of 2001. 10)
BMTKit	(As of 2001. 12)
SP/Cluster 1600 User's Guide	(As of 2002. 2)
eServer P690 DIY	(As of 2002. 5)
AIX PerfKit 2nd Edition (Performance Tuning Guide)	(As of 2002. 6)
LPAR + HACMP	(As of 2002. 6)
Porting Guide II - Porting Tip	(As of 2002. 12)
Migrating Solaris Applications to AIX - Quick Reference	(As of 2003. 6)