Linux System Programming #7-1
Lecture Notes

Spring 2020

School of Computer Science and Engineering,

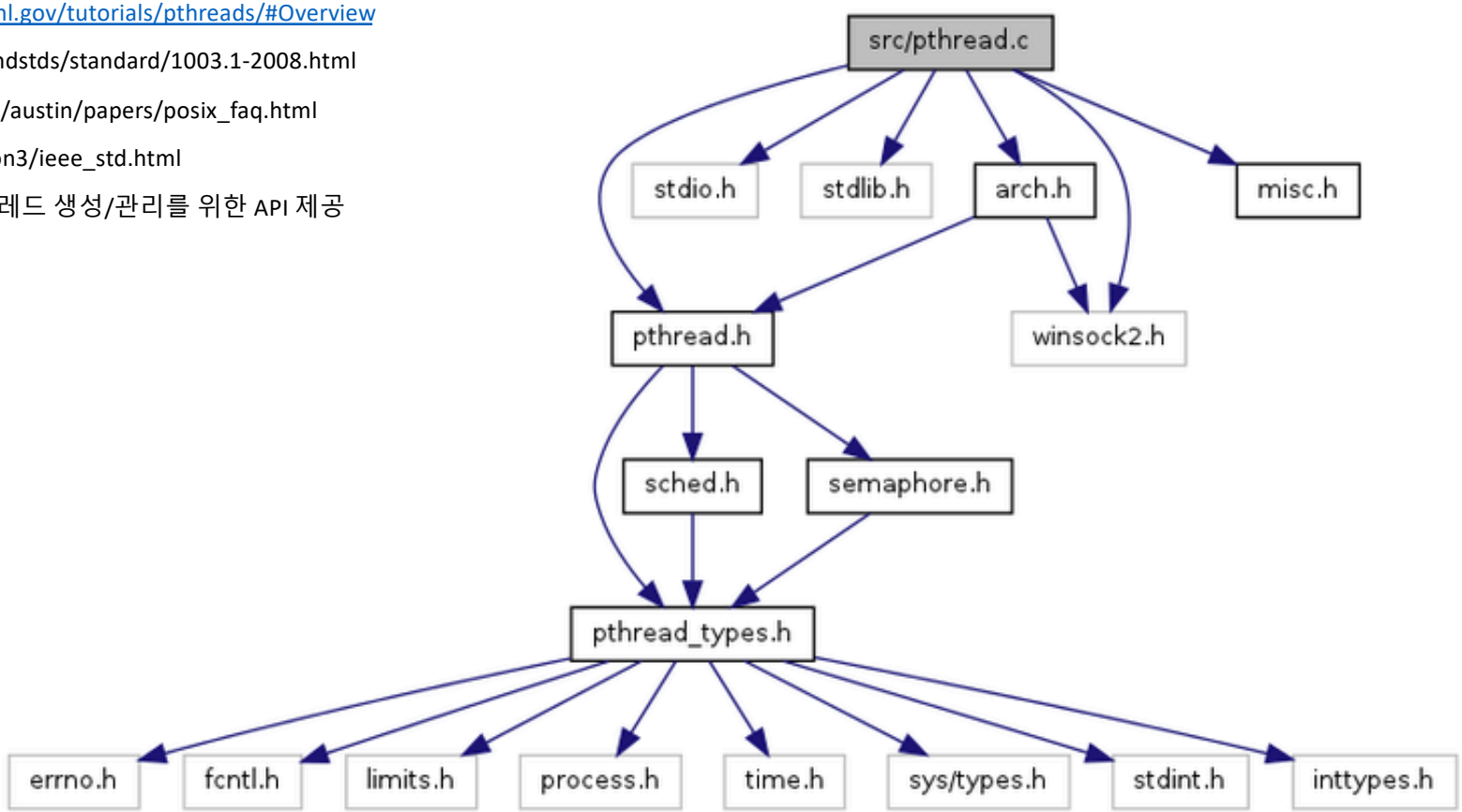
Soongsil University, Seoul, Korea

Jiman Hong

jiman@acm.org

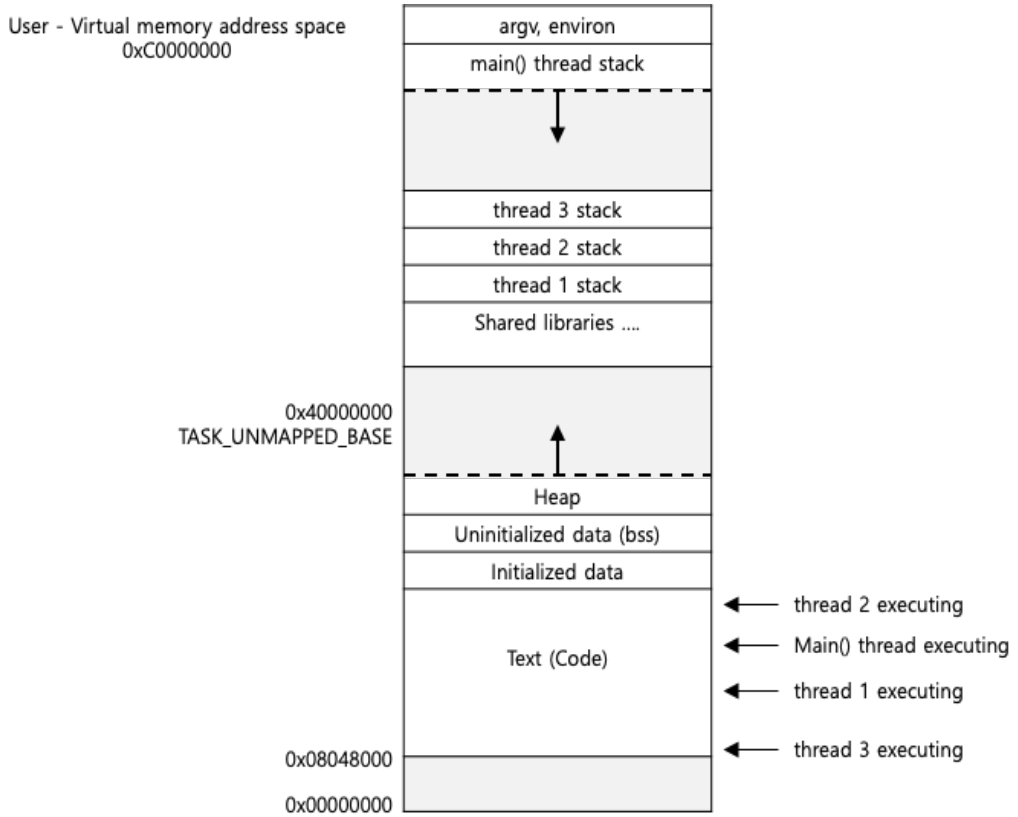
Pthread Lib.

- POSIX 1003.1 C (1995)
- https://computing.llnl.gov/tutorials/pthreads/#Overview
- standard.ieee.org/findstds/standard/1003.1-2008.html
- www.opengroup.org/austin/papers/posix_faq.html
- www.unix.org/version3/ieee_std.html
- 프로그래머에게 쓰레드 생성/관리를 위한 API 제공

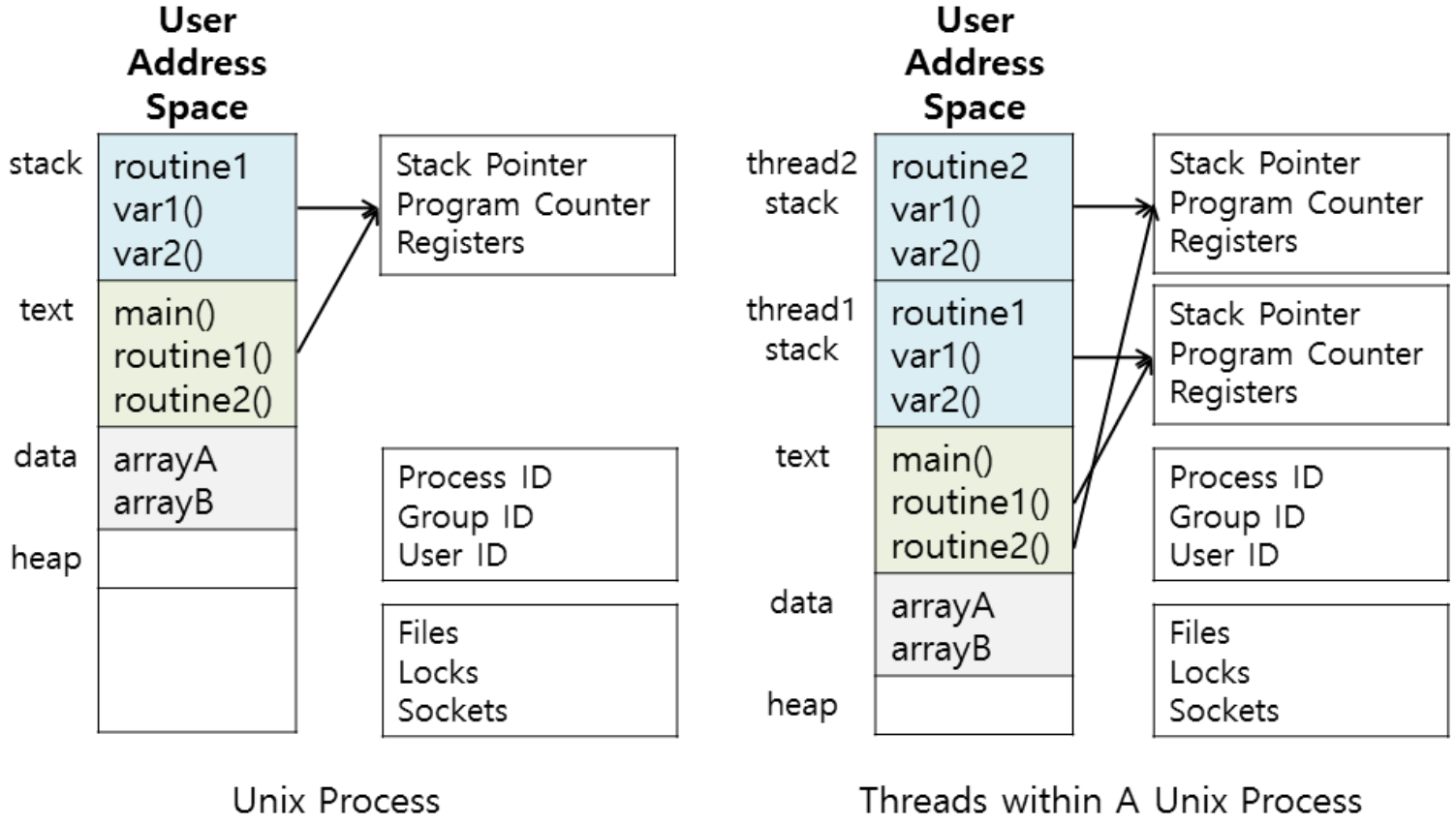


Include dependency graph for pthread.c:

단일 프로세스 내 다중 쓰레드의 가상 메모리 주소 공간



일반 프로세스(좌)와 쓰레드를 포함한 프로세스(우)



fork() vs pthread_create()

```c
//C Code for fork() creation test
#include <stdio.h>
#include <stdlib.h>
#define NFORKS 50000

void do_nothing() {
  int i;
   i = 0;
}

int main(int argc, char *argv[]) {
   int pid, j, status;
   for (j=0; j<NFORKS; j++) {
      if ((pid = fork()) < 0 ) {
         printf ("fork failed with error code= %d\n", pid);
         exit(0);
      }
     else if (pid ==0) {
         do_nothing();
         exit(0);
     }
      else {
         waitpid(pid, status, 0);
      }
  }
}
```

```c
//C Code for pthread() creation test
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NTHREADS 50000

void *do_nothing(void *null) {
  int i;
  i=0;
  pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
   int rc, i, j, detachstate;
   pthread_t tid;
   pthread_attr_t attr;
   pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
   for (j=0; j<NTHREADS; j++) {
      rc = pthread_create(&tid, &attr, do_nothing, NULL);
       if (rc) {
          printf("ERROR; return code from pthread_create() is %d\n", rc);
          exit(-1);
       }
     /* Wait for the thread */
      rc = pthread_join(tid, NULL);
      if (rc) {
           printf("ERROR; return code from pthread_join() is %d\n", rc);
           exit(-1);
      }
     }
    pthread_attr_destroy(&attr);
    pthread_exit(NULL);
}
```

fork() vs pthread_create()

- Timings reflect 50,000 process/thread creations, were performed with the time utility, and units are in seconds, no optimization flags.

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Compile

Compiler / Platform	Compiler Command	Description
INTEL Linux	**icc -pthread**	C
	icpc -pthread	C++
PGI Linux	**pgcc -lpthread**	C
	pgCC -lpthread	C++
GNU Linux, Blue Gene	**gcc -pthread**	GNU C
	g++ -pthread	GNU C++
IBM Blue Gene	**bgxlc_r / bgcc_r**	C (ANSI / non-ANSI)
	bgxlC_r, bgxlc++_r	C++

쓰레드 생성 및 종료

- pthread_create (thread,attr,start_routine,arg)
- pthread_exit (status)
- pthread_cancel (thread)
- pthread_attr_init (attr)
- pthread_attr_destroy (attr)

pthread_create()

```
#include <pthread.h>
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr,  void *(*start_rtn)(void *), void *restrict arg);
```
리턴 값: 성공 시 0, 실패 시 에러 번호

- 새로운 쓰레드를 생성하는 라이브러리 함수
- 쓰레드 프로그래밍에서 main()는 기본 쓰레드이며 추가적으로 쓰레드를 생성하길 원할 경우 명시적으로 프로그래머가 pthread_create()를 호출
- pthread_create() 호출 성공 : 새 쓰레드가 생성 -> 새 쓰레드의 쓰레드 ID는 tidp인자가 가리키는 주소에 저장
  - pthread_attr_t :여러 쓰레드 특성 설정, /usr/include/bits/pthreadtypes.h (linux), /usr/include/sys/types.h (solaris)
    - NULL => 기본 특성을 갖는 쓰레드가 생성
    - pthread_attr_init()로 pthread_attr_t 구조체 초기화
      - 사용자 모드 쓰레드 사용 : PTHREAD_ SCOPE_SYSTEM
      - 커널 모드 쓰레드 사용 : PTHREAD_SCOPE_ PROCESS
  - start_rtn : 새로 생성된 쓰레드는 start_rtn 함수의 주소에서 실행을 시작
    - start_rtn 함수는 형식이 없는 포인터인 arg를 인자로 받음

pthread_create()의 인자 및 설명

pthread_create() 인자	내용
thread	서브루틴에서 리턴되는 새로운 쓰레드의 유일한 식별자
attr	기본 값은 NULL이며, 쓰레드 애트리뷰터 객체 지정 시 사용. 즉, Linux에서는 NULL을 쓰면 사용자 모드(user mode) 쓰레드, Solaris와 같은 커널 쓰레드를 지원하는 OS에서는 커널 모드의 쓰레드 생성
start routine	쓰레드가 생성되면 해당 쓰레드가 실행될 C 루틴

pthread_create() 예제 1

<ssu_pthread_create_1.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <string.h>

void *ssu_thread(void *arg);

int main(void)
{
   pthread_t tid;
   pid_t pid;

   if (pthread_create(&tid, NULL, ssu_thread, NULL) != 0) {
      fprintf(stderr, "pthread_create error\n");
      exit(1);
   }

   pid = getpid();
   tid = pthread_self();
   printf("Main Thread: pid %u tid %u \n",
         (unsigned int)pid, (unsigned int)tid);
   sleep(1);
   exit(0);
}
```

```
void *ssu_thread(void *arg) {
   pthread_t tid;
   pid_t pid;

   pid = getpid();
   tid = pthread_self();
   printf("New Thread: pid %d tid %u \n", (int)pid, (unsigned int)tid);
   return NULL;
}
```

실행 결과

```
root@localhost:/home/oslab# gcc –o ssu_pthread_create_1
ssu_pthread_create_1.c -lpthread
root@localhost:/home/oslab# ./ssu_pthread_create_1
Main Thread: pid 3222 tid 3075864320
New Thread: pid 3222 tid 3075861312
```

pthread_create() 예제 2

359 page exam

pthread_exit(3)

```
#include <pthread.h>
void pthread_exit(void *rval_ptr);
```

- 쓰레드를 종료
- 쓰레드를 종료시키는 방법
  - (1) 시작 루틴에서 아무 일 없이 일을 수행하면 종료 후 리턴
  - (2) 쓰레드가 일을 완료하는지 여부에 상관없이 pthread_exit() 호출
  - (3) 같은 프로세스에서 어떤 쓰레드임에 상관없이 exec()나 exit()를 호출
  - (4) pthread_exit()를 명시적으로 호출하지 않았음에도 불구하고 main()가 종료되면 쓰레드는 종료
- 프로그래머가 종료 옵션 상태 파라미터를 직접 명세할 수 있게 하며, 종료 옵션 파라미터는 종료되는 쓰레드를 join 한 쓰레드에게 리턴
- 쓰레드 함수가 종료되었다고 모든 쓰레드의 자원이 종료되지는 않음
  - 종료된 쓰레드의 자원을 회수하기 위해 pthread_join()를 사용

pthread_exit() 예제 1

<ssu_pthread_exit_1.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define THREAD_NUM 5

void *ssu_printhello(void *arg);

int main(void)
{
   pthread_t tid[THREAD_NUM];
   int i;

   for (i = 0; i < THREAD_NUM; i++) {
      printf("In main: creating thread %d\n", i);

      if (pthread_create(&tid[i], NULL, ssu_printhello, (void *)&i) != 0) {
         fprintf(stderr, "pthread_create error\n");
         exit(1);
      }
   }

   pthread_exit(NULL);
   exit(0);
}
```

```
void *ssu_printhello(void *arg) {
   int thread_index;

   thread_index = *((int *)arg);
   printf("Hello World! It's me, thread #%d!\n", thread_index);
   pthread_exit(NULL);
   return NULL;
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_pthread_exit_1
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #1!
In main: creating thread 2
In main: creating thread 3
Hello World! It's me, thread #2!
Hello World! It's me, thread #3!
In main: creating thread 4
Hello World! It's me, thread #4!
Hello World! It's me, thread #5!
```

pthread_exit() 예제 2

<ssu_pthread_exit_2.c>

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void *ssu_thread(void *arg);

int main(void)
{
   pthread_t tid;

   if (pthread_create(&tid, NULL, ssu_thread, NULL) != 0) {
      fprintf(stderr, "pthread_create error\n");
      exit(1);
   }

   sleep(1);
   printf("쓰레드가 완료되기전 main 함수가 먼저 종료되면 실행중 쓰레드
소멸\n");
   printf("메인 종료\n");
   exit(0);
}

void *ssu_thread(void *arg) {
   printf("쓰레드 시작\n");
   sleep(5);
   printf("쓰레드 수행 완료\n");
   pthread_exit(NULL);
   return NULL;
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_pthread_exit_2
쓰레드 시작
쓰레드가 완료되기전 main 함수가 먼저 종료되면 실행중 쓰레드 소멸
메인 종료
```