Linux System Programming #7-2
Lecture Notes

Spring 2020

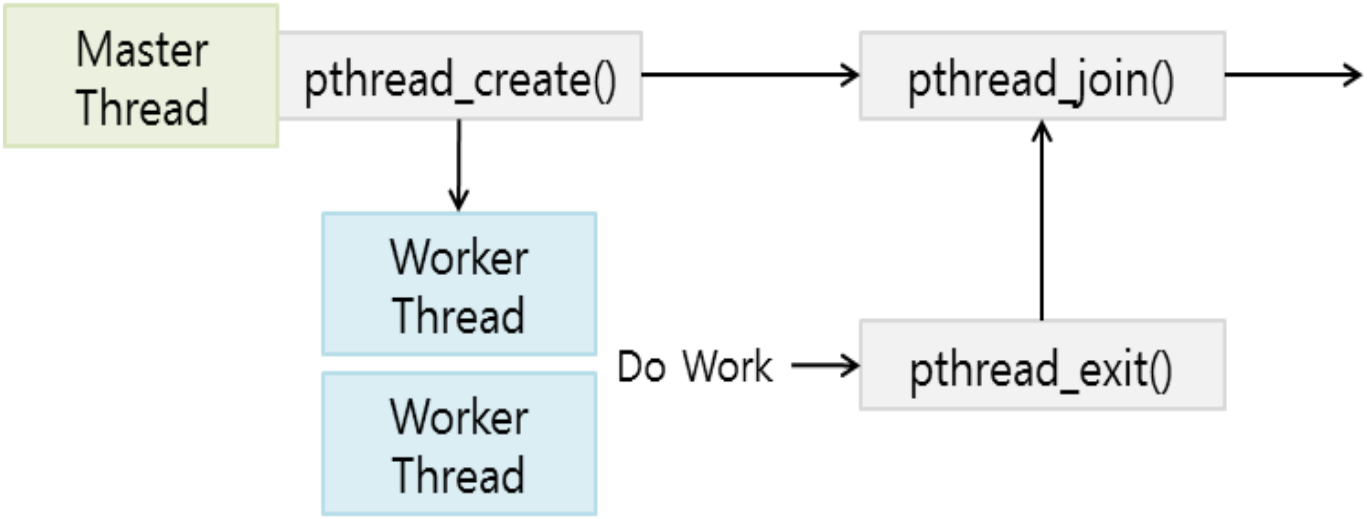
School of Computer Science and Engineering,
Soongsil University, Seoul, Korea
Jiman Hong
jiman@acm.org

pthread_join(3), pthread_detach(3)

```
#include <pthread.h>
int pthread_join(pthread_t thread, void **rval_ptr);
int pthread_detach(pthread_t tid);
```
리턴 값: 성공 시 0, 실패 시 에러 번호

- pthread_join()
  - main() 쓰레드가 생성한 쓰레드들이 종료될 때까지 기다리는 라이브러리 함수
  - main() 쓰레드의 종료로 인해 다른 쓰레드들이 강제로 종료되는 것을 방지하기 위해 다른 쓰레드들이 정상적으로 작업을 완료하고 종료될 수 있게 기다려주는 것
  - Thread : 종료 대기 및 리턴 값을 받을 쓰레드의 tid
    - pthread_create()를 통해 생성 된 tid 값을 인자로 넘겨줌
  - rval_ptr : 형식 없는 포인터(typeless pointer)로, 쓰레드 종료 루틴에 넘겨주는 인자와 동일한 인자
    - 쓰레드의 리턴 값을 사용하지 않는다면 rval_ptr 인자에 NULL 지정 당 쓰레드가 해당 종료 루틴에서 정상적으로 리턴되면 rval_ptr에는 그 루틴의 리턴 코드가 저장되고, 쓰레드가 취소되면 rval_ptr가 가리키는 곳에 PTHREAD_CANCELED가 설정
  - pthread_create()로 pthread_attr_t 인자를 NULL을 지정하여 기본 특성을 갖는 쓰레드를 생성할 경우 해당 쓰레드를 pthread_join 해야 함
    - 그렇지 않으면 쓰레드의 자원을 회수할 수 없게 되고 이로 인해 더 이상 쓰레드를 생성할 수 없게 됨
    - pthread_join()를 호출하면 해당 쓰레드는 자동으로 main() 쓰레드와 분리(detach) 상태가 되며, 커널은 해당 쓰레드가 종료 시 해당 쓰레드가 사용한 자원을 회수할 수 있게 됨
    - 쓰레드의 분리(detach) 상태 : 부모 쓰레드와 떨어져 독립적으로 동작한다는 뜻
- phread_detach()
  - 쓰레드 종료 시 자동으로 자원이 회수되도록 지정하는 라이브러리 함수
  - 한 쓰레드의 종료 상태는 그 쓰레드에 대한 pthread_join()가 호출될 때까지 유지됨
  - pthread_join()가 호출되어 분리된 쓰레드를 종료하는 경우에는 종료 즉시 그 쓰레드가 사용하던 자원들을 해제할 수 있지만, 그렇지 않고 쓰레드가 종료되면 pthread_join()로 그 쓰레드의 종료를 기다릴 수 없게 되기 때문에 쓰레드가 사용하였던 자원 회수 불가능

pthread_join() 와 pthread_exit()의 관계



pthread_exit() 예제 2

<ssu_pthread_join_1.c>

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

void *ssu_thread(void *arg);

int main(void)
{
   pthread_t tid1, tid2;
   int thread1 = 1;
   int thread2 = 2;
   int status;

   if (pthread_create(&tid1, NULL, ssu_thread, (void *)&thread1) != 0) {
      fprintf(stderr, "pthread_create error\n");
      exit(1);
   }

   if (pthread_create(&tid2, NULL, ssu_thread, (void *)&thread2) != 0) {
      fprintf(stderr, "pthread_create error\n");
      exit(1);
   }
   pthread_join(tid1, (void *)&status);
   pthread_join(tid2, (void *)&status);
   exit(0);
}
```

```
void *ssu_thread(void *arg) {
   int thread_index;
   int i;

   thread_index = *((int *)arg);

   for (i = 0; i < 5; i++) {
      printf("%d : %d\n", thread_index, i);
      sleep(1);
   }

   return NULL;
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_pthread_join_1
1 : 0
2 : 0
1 : 1
2 : 1
1 : 2
2 : 2
1 : 3
2 : 3
1 : 4
2 : 4
```

pthread_cancel(3), pthread_cleanup_push(3), pthread_cleanup_pop(3)

```
#include <pthread.h>
int pthread_cancel(pthread_t tid);
리턴 값: 성공 시 0, 실패 시 0이 아닌값을 리턴

void pthread_cleanup_push(void(*rtn)(void *), void *arg);
void pthread_cleanup_pop(int execute);
```

- pthread_cancel()
  - 지정한 쓰레드에 취소 요청 보내는 라이브러리 함수
  - pthread_cancel() = tid로 지정된 쓰레드가 PTHREAD_CANCELED를 인자로 하여 pthread_exit()를 호출한 것과 동일한 결과
  - 쓰레드가 취소 요청을 무시하거나 취소 시에 다른 행동이 일어나도록 설정한 경우 pthread_cancel()의 호출은 쓰레드가 종료될 때까지 기다리는 것이 아니고, 단순히 취소 요청만 할 뿐이기 때문에 쓰레드가 종료될 때까지 기다리지는 않는다
- pthread_cleanup_push(), pthread_cleanup_pop()
  - 종료 루틴 스택에 종료 루틴을 추가 및 삭제하는 라이브러리 함수
- pthread_cleanup_push()
  - 프로세스가 종료되는 경우 atexit()를 호출하여 사용자 함수를 등록하는 것처럼, 쓰레드가 종료되는 경우 호출 될 사용자 함수를 지정하는 역할
  - rtn으로 지정된 종료 루틴 함수를 스택에 등록하며, 등록된 종료 루틴 함수는 arg를 유일한 인자로 하여 호출
  - 종료 루틴 함수가 호출되는 상황은
    - (1) 쓰레드가 pthread_exit()를 호출하는 경우
    - (2) pthread_cancel()를 통해 취소 요청이 전달된 경우
    - (3) 쓰레드 내부에서 pthread_ cleanup_pop()를 execute 인자값에 0이 아닌 값을 넣어 호출한 경우

pthread_equal(3)

```
#include <pthread.h>
int pthread_equal(pthread_t t1, pthread_t t2);
```
리턴 값: t1과 t2가 같으면 0이 아닌 값, 다르면 0

- 인자로 주어진 두 개의 쓰레드 ID가 동일한지 확인하는 라이브러리 함수
- 비교 시 == 을 사용하지 않음

pthread_equal() 예제

```
<ssu_pthread_create_3.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *ssu_thread(void *arg);

pthread_t glo_tid;

int main(void)
{
   pthread_t loc_tid;

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

   sleep(5);

   if (pthread_equal(loc_tid, glo_tid) == 0) {
      printf("다른 쓰레드\n");
      exit(0);
   }

   printf("동일한 쓰레드\n");
   exit(0);
}

}
```

```
void *ssu_thread(void *arg) {
   printf("쓰레드에서 자신의 쓰레드 ID를 전역변수에 할당 \n");
   glo_tid = pthread_self();
   return NULL;
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_pthread_create_3
쓰레드에서 자신의 쓰레드 ID를 전역변수에 할당
동일한 쓰레드
```

pthread_self(3), pthread_once(3)

```
#include <pthread.h>
pthread_t pthread_self(void);
리턴 값: 호출한 쓰레드의 쓰레드 ID

int pthread_once(pthread_once_t *once_control, void (*init_routine)(void));  /* Ubuntu에는 존재하지 않음 */
리턴 값: 성공 시 0, 실패 시 에러 번호
```

- 현재 쓰레드의 아이디를 리턴하는 라이브러리 함수
  - 호출한 쓰레드의 ID(시스템에서 할당)를 리턴
- pthread_once()는 지정된 루틴이 한번만 효력을 갖게 만드는 라이브러리 함수
  - pthread_once()는 단 하나의 프로세스 내에서 init_routine(주로 초기화 루틴)을 실행
  - 특정 쓰레드에서 이 함수를 호출하면 다음의 호출들은 아무런 효력을 가지지 못함
  - once_control 인자는 동기화 구조체로 pthread_one()를 호출하기 전에 pthread_once_t once_control = PTHREAD_ONCE_INIT; 초기화 필요

pthread_self() 예제

```
<ssu_pthread_self.c>
#include <stdio.h>
#include <stdlib.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);
   }

   printf("%u\n", (unsigned int)tid);

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

   printf("%u\n", (unsigned int)tid);
   sleep(1);
   exit(0);
}
```

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

   tid = pthread_self();
   printf("->%u\n", (unsigned int)tid);
   return NULL;
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_pthread_self
3075693376
->3075693376
3067300672
->3067300672
```

Mutex

- Mutex

Thread1	Thread2	Balance
Read balance : $1,000		$1,000
	Read balance : $1,000	$1,000
	Deposit $200	$1,000
Deposit $200		$1,000
Update balance $1,000+$200		$1,200
	Update balance $1,000+$200	$1,200

pthread_mutex_init(3), pthread_mutex_destroy(3)

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexatt_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```
리턴 값: 성공 시 0, 실패 시 에러 번호

- pthread_mutex_init()
  - 뮤텍스 변수의 초기화
    - Pthread_mutex_init(&mutex_lock, NULL);
    - pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

- pthread_mutex_destroy()
  - 사용이 끝난 뮤텍스 변수 해제

pthread_mutex_lock() 예제

pthread_mutex_init(3), pthread_mutex_destroy(3)

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexatt_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```
리턴 값: 성공 시 0, 실패 시 에러 번호

- pthread_mutex_init()
  - 뮤텍스 변수의 초기화
    - Pthread_mutex_init(&mutex_lock, NULL);
    - pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
      - Protocol: Specifies the protocol used to prevent priority inversions for a mutex
      - Prioceiling: Specifies the priority ceiling of a mutex.
      - Process-shared: Specifies the process sharing of a mutex.
- pthread_mutex_destroy()
  - 사용이 끝난 뮤텍스 변수 해제

pthread_cond_init(3), pthread_ cond_destroy(3)

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *restirct condition, pthread_condattr_t *restrict attr);
int pthread_cond_destroy(pthread_cont_t *condition);
```
리턴 값: 성공 시 0, 실패 시 에러 번호

- 조건 변수의 초기화 및 해제를 위해 사용되는 라이브러리 함수
- 기존 뮤텍스는 사건을 기다리는 프로세스가 뮤텍스 변수를 지속적으로 검사해야 한다는 단점을 해결
  - 조건 변수를 사용하는 pthread_cond_init()와 pthread_cond_destroy() 사용
  - 조건 변수는 뮤텍스와 함께 사용되는 변수로 초기화 및 해제 방법이 뮤텍스와 유사

(참고) 기타 mutex 관련 함수

- int **pthread_cond_signal**(pthread_cond_t *cond);
  - cond 조건 변수에 신호를 보내 이 cond조건변수를 기다리고 있는 쓰레드 중의 하나를 다시 시작시킴
  - 만약 cond 조건 변수를 기다리고 있는 쓰레드가 없다면, 아무 일도 일어나지 않음
  - 만약 cond 조건 변수를 기다리는 쓰레드가 여러 개라면, 그 중의 하나만 깨어나지만, 특정 쓰레드를 지정할 수는 없음
- int **pthread_cond_broadcast**(pthread_cond_t *cond);
  - cond 조건 변수를 기다리고 있는 모든 쓰레드를 다시 시작시킴
  - 만약 cond 조건 변수를 기다리고 있는 쓰레드가 없다면, 아무 일도 일어나지 않음
- int **pthread_cond_wait**(pthread_cond_t *cond, pthread_mutex_t *mutex);
  - - 조건변수가 신호를 받을 때까지 기다리는 역할
- int **pthread_cond_timedwait**(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);
  - pthread_cond_wait()과 같은 역할을 하지만 기다릴 시간을 지정할 수 있다는 것이 다른 점
- int **pthread_cond_destroy**(pthread_cond_t *cond);
  - 조건변수를 삭제하고, 조건 변수에 할당된 자원해제

pthread_cond() 예제 1

pthread_cond() 예제 2

383

pthread_cond() 예제 3

385