2016년 2학기 **운영체제실습** 9주차 (2/2)

Thread

Dept. of Computer Engineering,
Kwangwoon Univ.

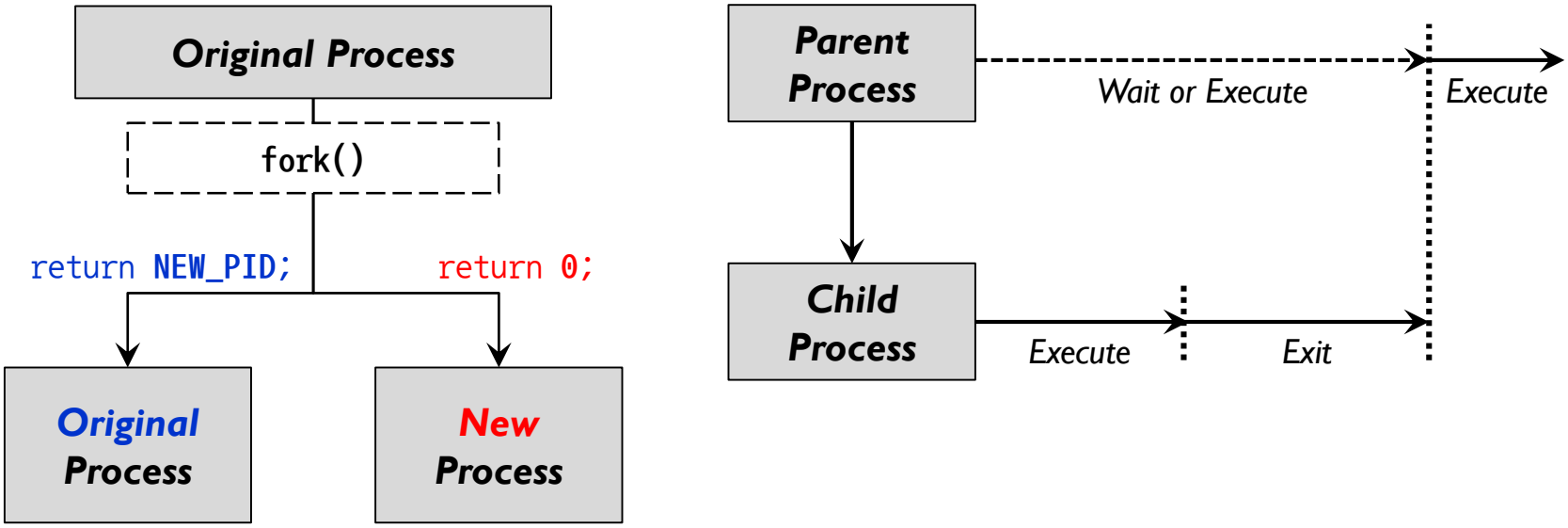
Contents

- Process Creation API
- 실습 1. Process Creation

- Thread의 이해
- POSIX Thread
- 실습 2. POSIX Thread

Process Creation API

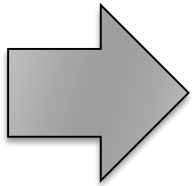
- fork()
  - 새로운 프로세스는 부모 프로세스로부터 생성
    - 생성된 프로세스 : 자식 프로세스 (child process)
    - fork()를 호출한 프로세스 : 부모 프로세스 (parent process)
    - 이 시점에서 두 프로세스가 동시 작업 수행



실습 1. Process Creation

```c
#include <stdio.h>
#include <sys/types.h>

#define MAX 5

void child();
void parent();

int main()
{
    pid_t pid;
    if( (pid = fork()) < 0 )
        return 1;
    else if( pid == 0 )
        child();
    else
        parent();
    return 0;
}

void child()
{
    int i;
    for( i=0 ; i<MAX ; ++i, sleep(1) )
        printf("child %d\n", i);
    printf("child done\n");
}

void parent()
{
    int i;
    for( i=0 ; i<MAX ; ++i, sleep(1) )
        printf("parent %d\n", i);
    printf("parent done\n");
}
```

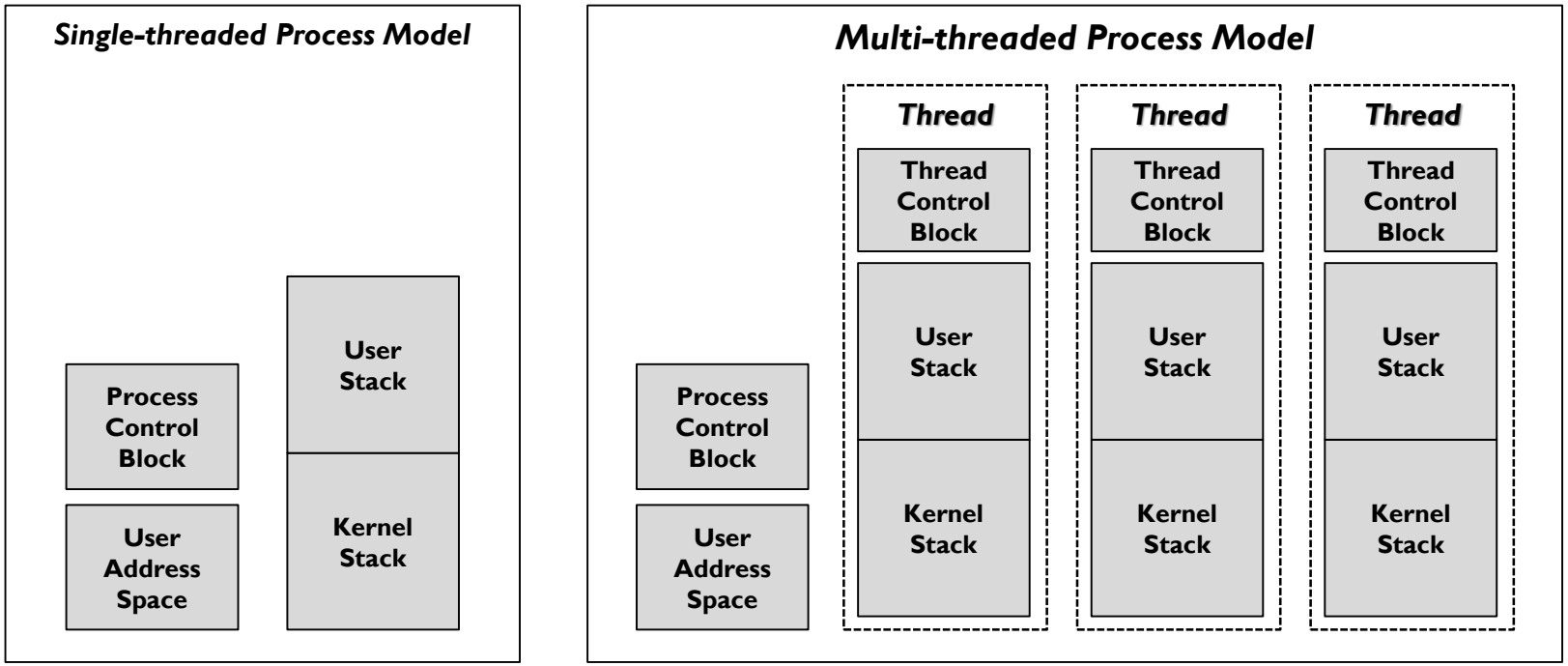
process.c



```
ssangkong@ssangkong-sslab:~/process$ make
cc    -c -o process.o process.c
cc   process.o   -o process
ssangkong@ssangkong-sslab:~/process$ ./process
parent 0
child 0
parent 1
child 1
child 2
parent 2
child 3
parent 3
child 4
parent 4
parent done
child done
```

Thread의 이해

- **Thread**
  - 특정 Process 내에서 실행되는 하나의 흐름을 나타내는 단위
  - 독립된 program counter를 갖는 단위
  - 독립된 register Set과 stack을 가짐
  - 비동기적인(asynchronous) 두 개의 작업이 서로 독립적으로 진행 가능
    - 처리를 위해 조건 변수나 mutex, semaphore와 같은 방법을 사용함



POSIX Thread

- **POSIX**
  - 이식 가능 운영 체제 인터페이스(Portable Operating System Interface)
  - 서로 다른 UNIX OS의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격

- **POSIX Thread**

함수명	설명
pthread_create	새로운 Thread를 생성함
pthread_detach	Thread가 자원을 해제하도록 설정
pthread_equal	두 Thread의 ID 비교
pthread_exit	Process는 유지하면서 지정된 Thread 종료
pthread_kill	해당 Thread에게 Signal을 보냄
pthread_join	임의의 Thread가 다른 Thread의 종료를 기다림
pthread_self	자신의 Thread id를 얻어옴

  - 컴파일시 `-lpthread` 혹은 `-pthread` 옵션 추가
    - e.g. `$ gcc -pthread thread_test.c`

POSIX Thread: Creation

- **Thread는 `pthread_t` 타입의 thread ID로 처리**
- **POSIX thread는 사용자가 지정한 특정 함수를 호출함으로써 시작**
  - 이 thread 시작 function은 void* 형의 인자를 하나 취한다

- **사용 함수: `pthread_create()`**

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void*), void *arg);
```

  - `pthread_t *thread` : Thread ID
  - `const pthread_attr_t *attr` : Thread 속성 지정. 기본값은 `NULL`.
  - `void *(*start)(void*)` : 특정 함수(**start**) 를 호출함으로써 thread가 시작
  - `void *arg` : start 함수의 인자

POSIX Thread: Termination

- **Process는 유지 하면서 `pthread_exit()` 함수를 호출하여 thread 자신을 종료**

- **단순히 thread를 종료 하는 역할만 수행**
  - 단, thread의 resource가 완전히 정리되지 않음

```
#include <pthread.h>

int pthread_exit(void *value_ptr);
```

  - `void *retval` : Return value가 저장. 사용하지 않으면, `NULL`

POSIX Thread: Detach and Join

- **Detach: 분리**
  - Process와 thread가 분리되면서 종료 시 자신이 사용했던 자원을 바로 반납
- **Join: 결합**
  - 생성된 thread가 `pthread_join()`을 호출한 thread에게 반환값을 전달하고 종료
- **즉, thread를 종료 할 때 분리 혹은 결합이 필요**

POSIX Thread: Detach

- **결합 가능(joinable)한 상태의 thread**
  - 분리되지 않은 thread
  - 종료되더라도 자원이 해제되지 않음

- **`pthread_detach()`**
  - Thread 종료 시 자원을 반납하도록 지정된 thread를 분리(detach) 상태로 만든다.

```
#include <pthread.h>

int pthread_detach (pthread_t thread);
```

    - Return value
      - 성공 시: 0
      - 실패 시: 0이 아닌 오류 코드

POSIX Thread: Join

- **다른 thread가 `thread_join()`을 반드시 호출해야 함**
  - Thead의 memory resource가 완전히 정리되지 않음

- **`pthread_join()`**
  - 지정된 thread가 종료될 때까지 호출 thread의 수행을 중단

```
#include <pthread.h>

int pthread_join (pthread_t thread, void **value_ptr);
```

  - `waitpid()`의 역할과 유사
  - `void **value_ptr` : thread의 종료코드가 저장될 장소

POSIX Thread: Thread Cleanup Handler

- **Thread cleanup handler 등록**
  - thread 종료 시 호출되는 특정 함수 등록
  - 하나의 thread에 둘 이상의 handler를 두는 것도 가능
    - 여러 handler는 하나의 스택에 등록

- **`pthread_cleanup_push()`**
  - 지정된 마무리 함수를 스택에 등록

```
#include <pthread.h>

void pthread_cleanup_push(void(*rtn)(void*), void* arg);
```

    - `rtn` : cleanup handler function

  - handler 호출 조건
    - thread가 `pthread_exit()` 호출
    - thread가 `pthread_cancel()`에 반응
    - thread가 execute 인수에 0이 아닌 값을 넣어 `pthread_cleanup_pop()`을 호출

POSIX Thread: Thread Cleanup Handler

- **Thread cleanup handler 제거**
  - 스택에 등록된 cleanup handler를 제거

- **`pthread_cleanup_pop()`**
  - 지정된 마무리 함수를 스택에서 제거

```
#include <pthread.h>

void pthread_cleanup_pop(int execute);
```

    - execute : 값이 0일 경우 등록된 handler를 호출하지 않음

  - cleanup handler는 스택에 등록된 반대 순서로 호출됨
  - 이들은 매크로로 구현될 수 있기 때문에, push-pop의 호출은 반드시 한 thread 범위 안에서 짝을 맞춰 주어야 함
    - push가 { 문자를 포함하고, pop이 } 문자를 포함

실습 2. POSIX Thread

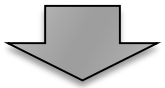
ssangkong — ssangkong@ssangkong-sslab: ~/thread — ssh — 80×53

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/types.h>
#include <linux/unistd.h>

void* thread_func(void *arg);
void cleanup_func(void *arg);
pid_t gettid(void);

int main()
{
    pthread_t tid[2];

    pthread_create(&tid[0], NULL, thread_func, (void*)0);
    pthread_create(&tid[1], NULL, thread_func, (void*)1);

    printf("main    gettid = %ld\n", (unsigned long)gettid());
    printf("main    getpid = %ld\n", (unsigned long)getpid());

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);

    return 0;
}

void* thread_func(void *arg)
{
    int i;
    pthread_cleanup_push(cleanup_func, "first cleanup");
    pthread_cleanup_push(cleanup_func, "second cleanup");
    printf("$tid[%d] start\n", (int)arg);
    printf("$tid[%d] gettid = %ld\n", (int)arg, (unsigned long)gettid());
    printf("$tid[%d] getpid = %ld\n", (int)arg, (unsigned long)getpid());
    for( i=0 ; i<0x40000000 ; ++i );
    if( (int)arg == 0 )
        pthread_exit(0);
    pthread_cleanup_pop(0);
    pthread_cleanup_pop(1);
    return (void*)1;
}

void cleanup_func(void *arg)
{
    printf("%s\n", (char*)arg);
}

pid_t gettid(void)
{
    return syscall(__NR_gettid);
}
```

thread.c 1,1 Top

:wq

```
LDFLAGS=-pthread

thread:thread.o

clean:
    $(RM) thread thread.o
```



```
ssangkong@ssangkong-sslab:~/thread$ make
cc    -c -o thread.o thread.c
cc -pthread  thread.o   -o thread
ssangkong@ssangkong-sslab:~/thread$ ./thread
main    gettid = 4933
main    getpid = 4933
$tid[0] start
$tid[0] gettid = 4934
$tid[0] getpid = 4933
$tid[1] start
$tid[1] gettid = 4935
$tid[1] getpid = 4933
^Z
[1]+  Stopped                 ./thread
ssangkong@ssangkong-sslab:~/thread$ ps -L
  PID   LWP TTY          TIME CMD
 3857  3857 pts/0    00:00:00 bash
 4933  4933 pts/0    00:00:00 thread
 4933  4934 pts/0    00:00:00 thread
 4933  4935 pts/0    00:00:00 thread
 4936  4936 pts/0    00:00:00 ps
ssangkong@ssangkong-sslab:~/thread$ fg
./thread
second cleanup
first cleanup
first cleanup
```

실습 2. POSIX Thread

```
1 LDFLAGS=-pthread
2
3 thread:thread.o
4
5 clean:
6     $(RM) thread thread.o
```

Linking시 자동으로 포함되는 변수

```
ssangkong@ssangkong-sslab:~/thread$ make
cc    -c -o thread.o thread.c
cc -pthread  thread.o   -o thread
ssangkong@ssangkong-sslab:~/thread$ ./thread
main    gettid = 4933
main    getpid = 4933
$tid[0] start
$tid[0] gettid = 4934
$tid[0] getpid = 4933
$tid[1] start
$tid[1] gettid = 4935
$tid[1] getpid = 4933
^Z
[1]+  Stopped                 ./thread
ssangkong@ssangkong-sslab:~/thread$ ps -L
  PID   LWP TTY          TIME CMD
 3857  3857 pts/0    00:00:00 bash
 4933  4933 pts/0    00:00:00 thread
 4933  4934 pts/0    00:00:00 thread
 4933  4935 pts/0    00:00:00 thread
 4936  4936 pts/0    00:00:00 ps
ssangkong@ssangkong-sslab:~/thread$ fg
./thread
second cleanup
first cleanup
first cleanup
```

Ctrl + z키를 누름. SIGSTOP Signal을 보냄.

LWP(Light-Weight Process) : Thread를 의미

fg명령어. SIGCONT Signal을 보냄.