KWANGWOON UNIVERSITY

2024년 2학기 **운영체제실습** 7주차

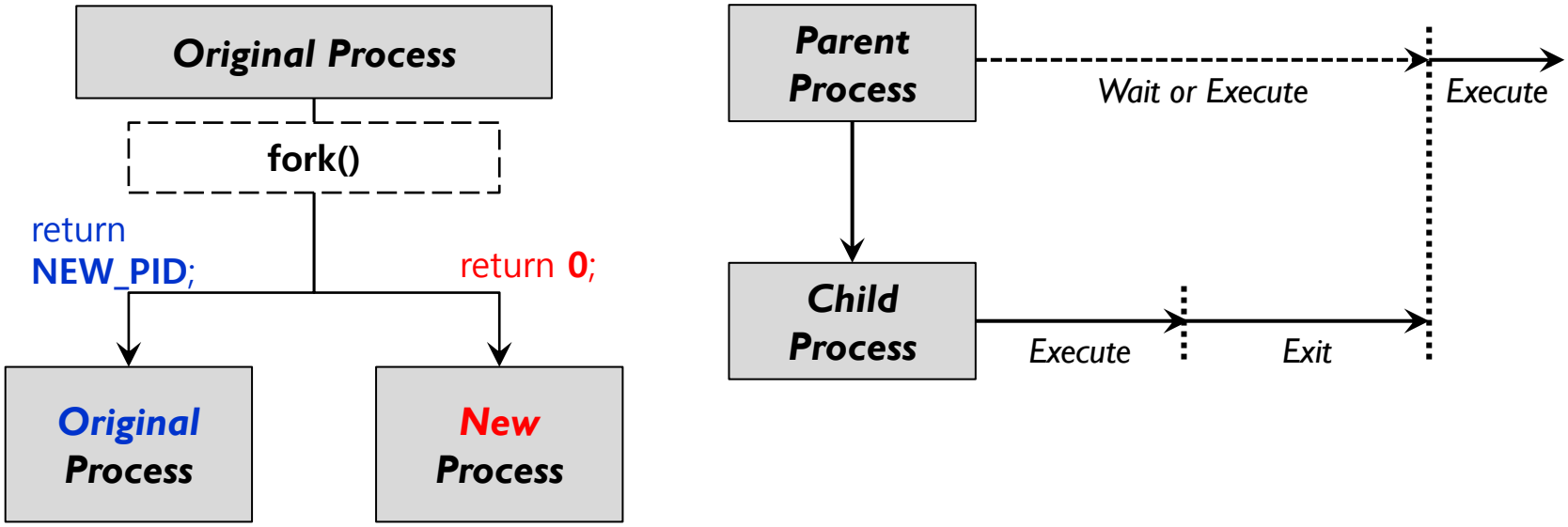
Thread

System Software Laboratory
School of Computer and Information Engineering
Kwangwoon Univ.

Contents

Process Creation API

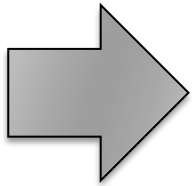
- **fork()**
  - 새로운 프로세스는 부모 프로세스로부터 생성
    - 생성된 프로세스 : 자식 프로세스 (child process)
    - fork()를 호출한 프로세스 : 부모 프로세스 (parent process)
    - 이 시점에서 두 프로세스가 동시 작업 수행



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



```
os2024123456@ubuntu:~/OS_prac/week7$ ./a.out
Parent 0
child 0
child 1
Parent 1
child 2
Parent 2
Parent 3
child 3
child 4
Parent 4
child done
Parent done
os2024123456@ubuntu:~/OS_prac/week7$ ./a.out
Parent 0
child 0
child 1
Parent 1
child 2
Parent 2
Parent 3
child 3
Parent 4
child 4
Parent done
child done
```

KWANGWOON UNIVERSITY

Process wait

- **자식 프로세스가 부모 프로세스에게 자원 반납**
- **좀비 프로세스를 만들지 않기 위함**
- 자식프로세스가 종료될 때까지 sleep

- **사용 함수: wait()**

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

  - int *status : status를 통해 자식 프로세스의 상태를 전달 받음
  - return : 종료된 프로세스의 pid

Process wait

- **특정 프로세스가 종료될 때까지 sleep**

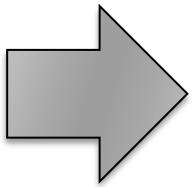
- **사용 함수: waitpid()**

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *status, int options);
```

  - pid_t pid : 종료한 프로세스의 pid
  - int *status : status를 통해 자식 프로세스의 상태를 전달 받음
  - int option : waitpid의 옵션
  - return : 종료된 프로세스의 pid

실습 2. Process Wait – wait( )

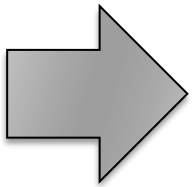
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <wait.h>

#define MAX 5

int main(){

pid_t pid;
int i, a;

for(i=0;i<MAX;i++)
{
        if( (pid = fork()) < 0)
                return 1;
        else if(pid == 0)
                exit(i);
}
for(i=0;i<MAX;i++){
        wait(&a);
        printf("origlnal exit valiable  : %d\n",a);
        printf("shift exit valiable     : %d\n\n",a>>8);
}
return 0;
}
```



```
os2024123456@ubuntu:~/OS_prac/week7$ ./a.out
Original exit variable  : 0
Shift exit variable     : 0

Original exit variable  : 256
Shift exit variable     : 1

Original exit variable  : 512
Shift exit variable     : 2

Original exit variable  : 768
Shift exit variable     : 3

Original exit variable  : 1024
Shift exit variable     : 4
```

실습 2. Process Wait – waitpid( )

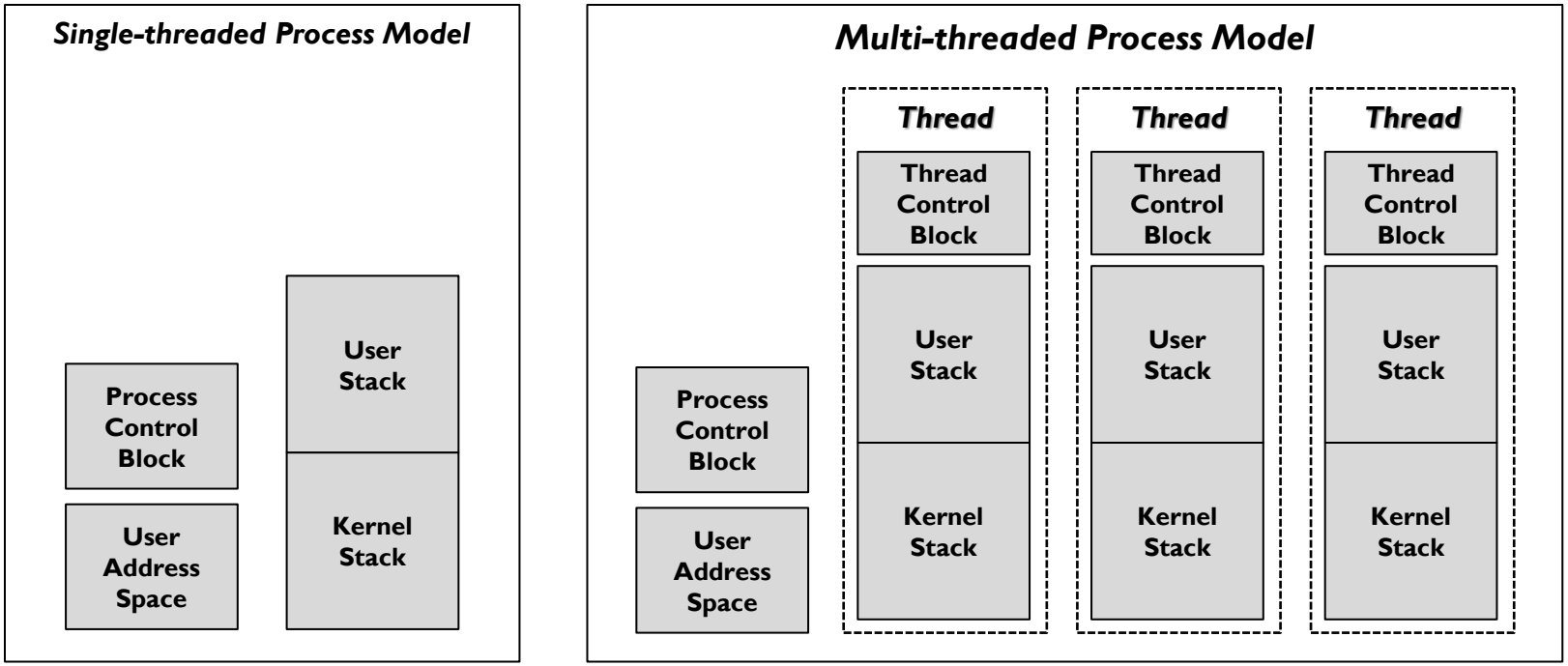
```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

void main(void)
{
    int pid1, pid2, status;
    pid_t child_pid;
    int a = 0;
    int b = 0;

    if ((pid1=fork()) == -1)
       perror("fork failed");

    if (pid1 == 0) {
        a = 1;
        printf("child_pid 1 = %d\n", getpid());
        if((pid2 = fork()) == 0){
                a = 2;
                printf("child_pid 2 = %d\n\n", getpid());
                exit(a);
        }
        else{
                child_pid = waitpid(pid2,&status,0);
                printf("child_pid : %d\n",child_pid);
                printf("original status : %d\n",status);
                printf("shift status : %d\n\n",status >> 8);
                exit(a);
        }
    }
    else if(pid1 != 0){
        child_pid = waitpid(pid1,&status,0);
        printf("child_pid : %d\n",child_pid);
        printf("original status : %d\n",status);
        printf("shift status : %d\n\n",status >> 8);
        }
}
```



```
os2019110613@ubuntu:~/test$ ./a.out
child_pid 1 = 9541
child_pid 2 = 9542

child_pid : 9542
original status : 512
shift status : 2

child_pid : 9541
original status : 256
shift status : 1
```

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 -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_routine)(void*), void *arg);
```

- pthread_t ***thread** : Thread ID
- const pthread_attr_t ***attr** : Thread 속성 지정. 기본값은 NULL.
- void *(***start_routine**)(void*) : 특정 함수(**start_routine**)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>

void pthread_exit(void *retval);
```

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

POSIX Thread: Detach and Join

- **Join: 결합**
  - 생성된 thread가 pthread_join()을 호출한 thread에게 반환값을 전달하고 종료

- **Detach: 분리**
  - Process와 thread가 분리되면서 종료 시 자신이 사용했던 **자원을 바로 반납**

- **즉, thread를 종료 할 때 결합 혹은 분리가 필요**

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 **retval);
```

  - waitpid()의 역할과 유사
  - void **retval : thread의 종료코드가 저장될 장소, 사용하지 않으면 NULL
  - Return value
    - 성공 시: 0
    - 실패 시: 0이 아닌 오류 코드

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);
```

  - pthread_t thread : 쓰레드 식별자 thread
  - Return value
    - 성공 시: 0
    - 실패 시: 0이 아닌 오류 코드

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(*routine)(void*), void* arg);
```

    - routine : cleanup handler function
    - void* arg: routine 함수의 인자
  - handler 호출 조건
    - thread가 pthread_exit() 호출
    - thread가 pthread_cancel()에 반응

```
int pthread_cancel(pthread_t thread);
```

      - 인자로 주어진 thread에 종료 요청을 보냄
    - 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 않고 삭제함
    값이 1일 경우 등록된 handler를 호출하고 삭제함

  - 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

```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;
}

```

실습 2. POSIX Thread(cont'd)

```
27 void* thread_func(void *arg)
28 {
29         int i;
30         pthread_cleanup_push(cleanup_func, "first cleanup");
31         pthread_cleanup_push(cleanup_func, "second cleanup");
32         printf("$tid[%d] start\n", (int)arg);
33         printf("$tid[%d] gettid = %ld\n", (int)arg, (unsigned long)gettid());
34         printf("$tid[%d] getpid = %ld\n", (int)arg, (unsigned long)getpid());
35
36         for( i=0; i<0x40000000; ++i);
37         if((int)arg == 0)
38         {
39                 pthread_exit(0);
40         }
41         pthread_cleanup_pop(0);
42         pthread_cleanup_pop(0);
43         return (void *)1;
44 }
45
46 void cleanup_func(void *arg)
47 {
48         printf("%s\n", (char *)arg);
49 }
50
51 pid_t gettid(void)
52 {
53         return syscall(__NR_gettid);
54 }
```

실습 2. POSIX Thread(cont'd)

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

```
os2024123456@ubuntu:~/OS_prac/week7$ ./thread
main    gettid = 12537
main    getpid = 12537
$tid[0] start
$tid[0] gettid = 12538
$tid[0] getpid = 12537
$tid[1] start
$tid[1] gettid = 12539
$tid[1] getpid = 12537
second cleanup
first cleanup
```

실습 3. POSIX Thread

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

Linking시 자동으로 포함되는 변수

```
os2024123456@ubuntu:~/OS_prac/week7$ ./thread
$tid[0] start
$tid[0] gettid = 12193
$tid[0] getpid = 12192
$tid[1] start
$tid[1] gettid = 12194
$tid[1] getpid = 12192
main    gettid = 12192
main    getpid = 12192
^Z
[1]+  Stopped                 ./thread
os2024123456@ubuntu:~/OS_prac/week7$ ps -L
    PID     LWP TTY          TIME CMD
   2009    2009 pts/0    00:00:00 bash
  12192   12192 pts/0    00:00:00 thread
  12192   12193 pts/0    00:00:01 thread
  12192   12194 pts/0    00:00:01 thread
  12195   12195 pts/0    00:00:00 ps
os2024123456@ubuntu:~/OS_prac/week7$ fg
./thread
second cleanup
first cleanup
```

Ctrl + z키를 누름. SIGSTOP Signal을 보냄.

LWP(Light-Weight Process) : Thread를 의미

fg명령어. SIGCONT Signal을 보냄.

KWANGWOON UNIVERSITY