



(Operating System) Practice -5-

Make Macro & Fork



Index

- I. Make Macro
- II. Fork

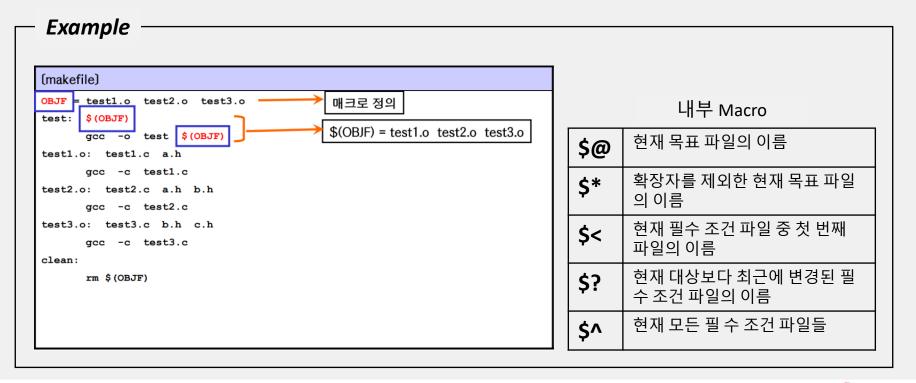




Make Macro

Make Macro

- makefile 작성 시, 동일한 파일의 이름을 반복해서 적는 비효율 발생
- 이런 비효율을 줄이기 위해 Make의 macro 기능을 활용할 수 있음
- 반복되는 파일들의 이름을 하나로 묶어서 정의 가능
- 내부 macro 사용 가능





Make Macro

Make Macro

- 예제

```
1 #include <stdio.h> make_test2.c
2
3 extern void func1(){
4 printf("test2.c has run.\n");
5 }
```

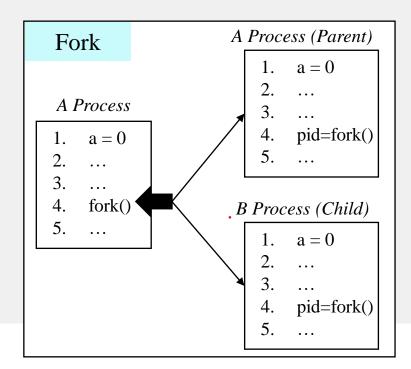
```
#include <stdio.h> make_test3.c

extern void func2(){{
    printf("test3.c has run.\n");
}
```



Fork

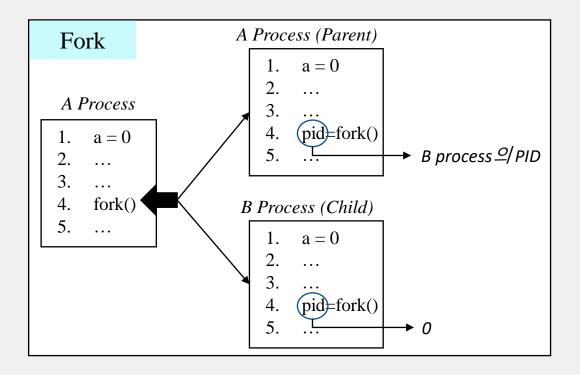
- Unix 계열 운영체제에서 제공하는 표준 프로세스 생성 함수
- Fork의 기능을 한 마디로 얘기하자면 "자기 자신을 복제" 하는 것
- Fork 기능을 호출한 프로세스와 완전히 동일한 프로세스를 메모리에 할당하여 실행함
 - ✓ 원본 프로세스 → 부모 프로세스, 복제된 프로세스 → 자식 프로세스
 - ✓ 부모 프로세스와 자식 프로세스는 각기 다른 메모리 공간을 갖음
- 멀티 프로세스 동작을 수행할 수 있게 함





Fork

- 부모 프로세스에게는 자식 프로세스의 Process ID(PID)가 리턴 됨
- 자식 프로세스에게는 0이 리턴 됨





• Fork 함수

fork()		
기능	자식 프로세스 생성	
기본형	pid_t fork(void)	
반환 값	부모 프로세스: 자식 프로세스의 ID 자식 프로세스: 0 생성 실패: -1	
헤더 파일	<sys type.h=""> <unistd.h></unistd.h></sys>	

exit()		
기능	정상적으로 프로세스 종료	
기본형	void exit(int status)	
반환 값	없음	
헤더 파일	<stdlib.h></stdlib.h>	



• Fork 예제

```
#include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include <string.h>
    int g_var = 1;
    void main(){
         int l var = 10;
         char name[10];
         int pid;
11
        printf("Fork starts\n");
         pid = fork();
         if(pid<0){
             printf("fork failed\n");
         else if(pid==0){ // child
             strcpy(name, "child");
             g var++;
             l var++;
         else{ // prant
             sleep(3);
             strcpy(name, "parent");
         printf("pid=%d, name=%s, g var=%d,l var=%d\n", getpid(), name, g var, l var);
         exit(0);
```



Orphan Process & Zombie Process

- Orphan Process
 - ✓ 부모 프로세스가 자식 프로세스보다 먼저 종료되는 경우
 - ✓ 부모 프로세스의 PID를 확인할 수 없게 됨 → 프로그램의 오류를 야기하기도 함
- Zombie Process
 - ✓ 자식 프로세스가 부모 프로세스보다 먼저 종료되는 경우
 - ✓ 부모 프로세스는 종료된 자식 프로세스의 일부 정보(PID, 종료 상태, 커널에서 사용하는 구조체 등)를 유지함
 - ✓ 메모리 낭비를 초래하고, PID를 차지 하기 때문에 다른 프로세스의 실행을 방해할 수 있음
- Orphan Process와 Zombie Process 모두 wait() 함수를 통해 해결될 수 있음

wait()		
기능	종료된 자식 프로세스의 반환 값 요청, 자식 프로세스가 종료될 때까지 기다림 호출 시점에 종료된 자식 프로세스가 없으면 blocking 상태에 빠짐 (무한 실행 중)	
기본형	pid_t wait(int * status) status: 자식 프로세스가 종료될 때의 상태 정보	
반환 값	성공: 종료된 자식 프로세스 ID 실패: -1	
헤더 파일	<sys type.h=""> <sys wait.h=""></sys></sys>	



Orphan Process 예제

- Orphan Process 문제를 야기하는 Code

```
1 v #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    void main(){
        int pid;
        pid = fork();
        if(pid == 0){
            sleep(1); // This line results in orphan process.
            printf("----\n");
11
            printf("This is child process.\n PID is %d\n praent's PID is %d\n", getpid(), getppid());
12
            exit(0);
13
14
        else{
15
            printf("----\n");
            printf("This is parent process.\n PID is %d\n child's PID is %d\n", getpid(), pid);
17
19
```



• Orphan Process 예제

- Orphan Process 문제를 해결한 Code

```
#include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include <sys/wait.h>
    void main(){
        int pid;
        int status; .
        int terminatedPid;
        pid = fork();
        if(pid ==0){
            sleep(1); // This line results in orphan process.
            printf("----\n");
            printf("This is child process.\n PID is %d\n praent's PID is %d\n", getpid(), getppid());
15
            exit(0);
        else{
            printf("----\n");
19
            printf("This is parent process.\n PID is %d\n child's PID is %d\n", getpid(), pid);
21
22
            terminatedPid = wait(&status);
23
            printf("----\n");
            printf("parent waited for %d, and %d's status is %d\n", terminatedPid, terminatedPid, status);
```



• Zombie Process 예제

- Zombie Process 문제를 야기하는 Code
- Zombie Process 확인 방법
 - ✓ Background로 예제코드 실행 \rightarrow ./zombie_process & OR ./zombie_process \rightarrow "ctrl+z" \rightarrow bg
 - ✓ Zombie Process 확인 → ps -ef | grep defunct | grep -v grep

```
#include <stdio.h>
     #include <unistd.h>
     #include <stdlib.h>
     void main(){
         int pid;
         pid = fork();
         if(pid==0){
             printf("\nChild process(%d) is created and exits now.\n", getpid());
10
             exit(0);
11
12
13
         printf("\nParent process(%d) is going to sleep.\n", getpid());
         sleep(60);
14
         printf("\nParent process exits...\n");
15
```



• Zombie Process 예제

- Zombie Process 문제를 해결한 Code

```
#include <stdio.h>
    #include <unistd.h>
    #include <stdlib.h>
    #include <sys/wait.h>
     void main(){
         int pid;
         pid = fork();
         if(pid==0){
10
             printf("\nChild process(%d) is created and exits now.\n", getpid());
11
             exit(0);
12
13
         printf("\nParent process(%d) is going to sleep.\n", getpid());
14
         wait(NULL);
15
         sleep(60);
16
         printf("\nParent process exits...\n");
17
18
```

