

(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

Example

```
[makefile]
OBJF = test1.o test2.o test3.o
test: $(OBJF)
    gcc -o test $(OBJF)
test1.o: test1.c a.h
    gcc -c test1.c
test2.o: test2.c a.h b.h
    gcc -c test2.c
test3.o: test3.c b.h c.h
    gcc -c test3.c
clean:
    rm $(OBJF)
```

매크로 정의

\$(OBJF) = test1.o test2.o test3.o

내부 Macro

| | |
|-----|------------------------------|
| \$@ | 현재 목표 파일의 이름 |
| \$* | 확장자를 제외한 현재 목표 파일의 이름 |
| \$< | 현재 필수 조건 파일 중 첫 번째 파일의 이름 |
| \$? | 현재 대상보다 최근에 변경된 필수 조건 파일의 이름 |
| ^ | 현재 모든 필수 조건 파일들 |

Make Macro

- Make Macro

- 예제

```
1  #include <stdio.h>
2
3  void func1();
4  void func2();
5
6  int main(){
7      printf("test1.c has run.\n");
8      func1();
9      func2();
10 }
```

make_test1.c

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

make_test2.c

```
1  #include <stdio.h>
2
3  extern void func2(){
4      printf("test3.c has run.\n");
5  }
```

make_test3.c

```
1  OBJF = make test1.o make test2.o make test3.o
2
3  test : $(OBJF)
4      gcc $(OBJF) -o $@
5  make_test1.o : make_test1.c
6      gcc -c make_test1.c
7  make_test2.o : make_test2.c
8      gcc -c make_test2.c
9  make_test3.o : make_test3.c
10     gcc -c make_test3.c
11  clean :
12     rm $(OBJF)
```

makefile

Fork

• 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 가짐
- 멀티 프로세스 동작을 수행할 수 있게 함



Fork

- Fork

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



Fork

• 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/types.h> <unistd.h> |

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

Fork

- Fork 예제

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <string.h>
5
6  int g_var = 1;
7
8  void main(){
9      int l_var = 10;
10     char name[10];
11     int pid;
12
13     printf("Fork starts\n");
14
15     pid = fork();
16     if(pid<0){
17         printf("fork failed\n");
18     }
19     else if(pid==0){ // child
20         strcpy(name,"child");
21         g_var++;
22         l_var++;
23     }
24     else{ // parent
25         sleep(3);
26         strcpy(name,"parent");
27     }
28     printf("pid=%d, name=%s, g_var=%d, l_var=%d\n", getpid(), name, g_var, l_var);
29     exit(0);
30 }
```


Fork

• 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> |

Fork

- Orphan Process 예제

- Orphan Process 문제를 야기하는 Code

```
1  ✓ #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  void main(){
6      int pid;
7      pid = fork();
8
9      if(pid ==0){
10         sleep(1); // This line results in orphan process.
11         printf("-----\n");
12         printf("This is child process.\n PID is %d\n parent's PID is %d\n", getpid(), getppid());
13         exit(0);
14     }
15     else{
16         printf("-----\n");
17         printf("This is parent process.\n PID is %d\n child's PID is %d\n", getpid(), pid);
18     }
19 }
```

Fork

- Orphan Process 예제

- Orphan Process 문제를 해결한 Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  void main(){
7      int pid;
8      int status;
9      int terminatedPid;
10     pid = fork();
11
12     if(pid == 0){
13         sleep(1); // This line results in orphan process.
14         printf("-----\n");
15         printf("This is child process.\n PID is %d\n parent's PID is %d\n", getpid(), getppid());
16         exit(0);
17     }
18     else{
19         printf("-----\n");
20         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");
24         printf("parent waited for %d, and %d's status is %d\n", terminatedPid, terminatedPid, status);
25     }
26 }
```

Fork

• Zombie Process 예제

– Zombie Process 문제를 야기하는 Code

– Zombie Process 확인 방법

✓ Background로 예제코드 실행 → `./zombie_process &` OR `./zombie_process` → "ctrl+z" → `bg`

✓ Zombie Process 확인 → `ps -ef | grep defunct | grep -v grep`

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4
5  void main(){
6      int pid;
7      pid = fork();
8
9      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     sleep(60);
15     printf("\nParent process exits...\n");
```

Fork

- **Zombie Process 예제**

- Zombie Process 문제를 해결한 Code

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <sys/wait.h>
5
6  void main(){
7      int pid;
8      pid = fork();
9
10     if(pid==0){
11         printf("\nChild process(%d) is created and exits now.\n", getpid());
12         exit(0);
13     }
14     printf("\nParent process(%d) is going to sleep.\n", getpid());
15     wait(NULL);
16     sleep(60);
17     printf("\nParent process exits...\n");
18 }
```