6주차 3차시 프로세스 동기화

【학습목표】

1. 프로세스 동기화에 대해 설명할 수 있다.
2. 프로세스 동기화 함수를 사용할 수 있다.

학습내용1 : 프로세스 동기화

1. 부모 프로세스와 자식 프로세스의 종료절차

부모 프로세스와 자식 프로세스는 순서와 상관없이 실행하고 먼저 실행을 마친 프로세스는 종료
부모 프로세스와 자식 프로세스 사이에 종료절차가 제대로 진행되지 않으면 좀비 프로세스 발생

2. 동기화 필요 이유

① 좀비프로세스 종료
실행을 종료하고 자원을 반납한 자식 프로세스의 종료 상태를 부모 프로세스가 가져가지 않으면 좀비 프로세스 발생
좀비 프로세스는 프로세스 테이블에만 존재
좀비 프로세스는 일반적인 제거 방법은 없음
좀비 프로세스를 방지하기 위해 부모 프로세스와 자식 프로세스를 동기화 해야함

② 고아프로세스 종료
자식 프로세스보다 부모 프로세스가 먼저 종료할 경우 자식 프로세스들은 고아 프로세스가 됨
고아 프로세스는 1번 프로세스(init)의 자식 프로세스로 등록

학습내용2 : 프로세스 동기화 함수

1. 프로세스 동기화: wait(3)

- wait 함수는 자식 프로세스가 종료할 때까지 부모 프로세스를 기다리게 함
- 부모 프로세스가 wait 함수를 호출하기 전에 자식 프로세스가 종료하면 wait 함수는 즉시 리턴
- wait 함수의 리턴값은 자식 프로세스의 PID
- 리턴값이 -1이면 살아있는 자식 프로세스가 하나도 없다는 의미

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *stat_loc);
```

stat_loc : 상태정보를 저장할 주소

① wait 함수 사용하기

```
...
07  int main(void) {
08      int status;
09      pid_t pid;
11      switch (pid = fork()) {
12          case -1 :  /* fork failed */
13              perror("fork");
14              exit(1);
15              break;
16          case 0 :   /* child process */
17              printf("--> Child Process\n");
18              exit(2);
19              break;
20          default :   /* parent process */
21              while (wait(&status) != pid)
22                  continue;
23              printf("--> Parent process\n");
24              printf("Status: %d, %x\n", status, status);
25              printf("Child process Exit Status:%d\n",status >> 8);
26              break;
27      }
29      return 0;
30  }
```

```
# ex6_8.out
--> Child Process
--> Parent process
Status: 512, 200
Child process Exit Status:2
```

자식 프로세스의 종료를 기다림

오른쪽으로 8비트 이동해야 종료 상태값을 알 수 있음

2. 특정 자식 프로세스와 동기화: waitpid(3)

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

* pid에 지정할 수 있는 값
-1보다 작은 경우 : pid의 절댓값과 같은 프로세스 그룹ID에 속한 자식 프로세스 중 임의의 프로세스의 상태값 요청
-1인 경우 : wait 함수처럼 임의의 자식 프로세스의 상태값을 요청
0인 경우 : 함수를 호출한 프로세스와 같은 프로세스 그룹에 속한 임의의 프로세스의 상태값 요청
0보다 큰 경우 : 지정한 PID의 상태값 요청

* options: waitpid 함수의 리턴 조건
WCONTINUED: 수행중인 자식 프로세스의 상태값 리턴
WNOHANG: pid로 지정한 자식프로세스의 상태값을 즉시 리턴받을 수 없어도 이를 호출한 프로세스의 실행을 블록하지 않고 다른 작업을 수행토록 함
WNOWAIT: 상태값을 리턴한 프로세스가 대기 상태에 머물 수 있도록 함
WUNTRACED: 실행을 중단한 자식 프로세스의 상태값을 리턴

* waitpid 함수 사용하기

```
...
07  int main(void) {
08      int status;
09      pid_t pid;
10
11      if ((pid = fork()) < 0) { /* fork failed */
12          perror("fork");
13          exit(1);
14      }
15
16      if (pid == 0) { /* child process */
17          printf("--> Child process\n");
18          sleep(3);
19          exit(3);
20      }
21
22      printf("--> Parent process\n");
23
24      while (waitpid(pid, &status, WNOHANG) == 0) {
25          printf("Parent still wait...\n");
26          sleep(1);
27      }
28
29      printf("Child Exit Status : %d\n", status>>8);
30
31      return 0;
32  }
```

```
# ex6_9.out
--> Child process
--> Parent process
Parent still wait...
Parent still wait...
Parent still wait...
Child Exit Status : 3
```

WNOHANG이므로 waitpid 함수는 블록되지 않고 25~26행 반복 실행

【학습정리】

1. 프로세스 동기화
- 부모 프로세스와 자식 프로세스는 순서와 상관없이 실행하고 먼저 실행을 마친 프로세스는 종료
- 부모 프로세스와 자식 프로세스 사이에 종료절차가 제대로 진행되지 않으면 좀비 프로세스 발생

2. 동기화 필요 이유
- 좀비프로세스 종료
- 고아프로세스 종료