6 프로세스 생성과 실행

학습목표

- 프로세스를 생성하는 방법을 이해한다.
- 프로세스를 종료하는 방법을 이해한다.
- exec함수군으로 새로운 프로그램을 실행하는 방법을 이해한다.
- 프로세스를 동기화하는 방법을 이해한다.

동기화: 프로세스가 다른 프로세스를 기다리면서 무언가를 처리함

목차

- 프로세스 생성
- 프로세스 종료함수
- exec 함수군 활용
- exec 함수군과 fork 함수
- 프로세스 동기화

프로세스 생성[1]

프로그램 실행 : system(3)

```
#include <stdlib.h>
int system(const char *string);
```

- 새로운 프로그램을 실행하는 가장 간단한 방법이나 비효율적이므로 남용하지 말 것
- 실행할 프로그램명을 인자로 지정

[예제 6-1] system 함수 사용하기 ex6_1.c

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      int a;
06      a = system("ps -ef | grep han > han.txt");
07      printf("Return Value : %d\n", a);
08
09      return 0;
10  }
```

han 이라는 키워드를 찾아내서 han.txt 라는 파일 만들고 거기에 다 저장

```
# ex6_1.out
Return Value : 0
```

```
# cat han.txt
root 736 735 0 10:31:02 pts/3  0:00 grep han
root 735 734 0 10:31:02 pts/3  0:00 sh -c ps -ef | grep han> han.txt
```

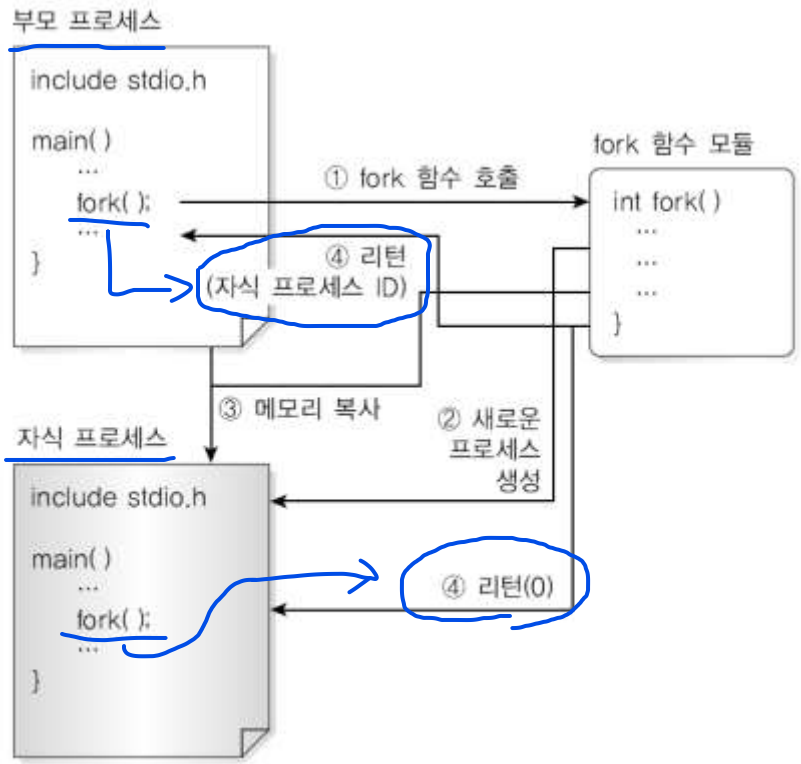
프로세스 생성[2]

프로세스 생성: fork(2)

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

- 새로운 프로세스를 생성 : 자식 프로세스
- fork 함수를 호출한 프로세스 : 부모 프로세스
- 자식 프로세스는 부모 프로세스의 메모리를 복사
  - RUID, EUID, RGID, EGID, 환경변수
  - 열린 파일기술자, 시그널 처리, setuid, setgid
  - 현재 작업 디렉토리, umask, 사용가능자원 제한

- 부모 프로세스와 다른 점
  - 자식 프로세스는 유일한 PID를 갖는다
  - 자식 프로세스는 유일한 PPID를 갖는다.
  - 부모 프로세스가 설정한 프로세스잠금, 파일 잠금, 기타 메모리 잠금은 상속 안함
  - 자식 프로세스의 tms구조체 값은 0으로 설정 (시간)
- 부모 프로세스와 자식 프로세스는 열린 파일을 공유하므로 읽거나 쓸 때 주의해야 한다.



[그림 6-1] fork 함수를 이용한 새로운 프로세스 생성

fork 하기전 부모가 open했던 파일을 자식이 사용가능 (fork후는 X)

[예제 6-2] fork 함수 사용하기

ex6_2.c

```
...
06  int main(void) {
07      pid_t pid;
08
09      switch (pid = fork()) {
10          case -1 :  /* fork failed */
11              perror("fork");
12              exit(1);
13              break;
14          case 0 :   /* child process */
15          printf("Child Process - My PID:%d, My Parent's PID:%d\n",
16                  (int)getpid(), (int)getppid());
17              break;
18          default : /* parent process */
19          printf("Parent process - My PID:%d, My Parent's PID:%d, "
                "My Child's PID:%d\n", (int)getpid(), (int)getppid(),
                (int)pid);
21              break;
22      }
23
24      pri
26      ret
27  }
```

fork함수의 리턴값 0은 자식 프로세스가 실행

부모는 fork 값이 자식의 pid

```
# ex6_2.out
Child Process - My PID:796, My Parent's PID:795
End of fork
Parent process - My PID:795, My Parent's PID:695, My Child's PID:796
End of fork
```

프로세스 종료 함수[1]

- 프로그램 종료: exit(2)

```
#include <stdlib.h>
void exit(int status);
```

  - status : 종료 상태값 (부모에게 전달됨)

- 프로그램 종료시 수행할 작업 예약: atexit(2)

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

  - func : 종료시 수행할 작업을 지정한 함수명

- 프로그램 종료: _exit(2)

```
#include <unistd.h>
void _exit(int status);
```

  - 일반적으로 프로그램에서 직접 사용하지 않고 exit 함수 내부적으로 호출

쓸일없다는 소리!

프로세스 종료 함수[2]

- 프로그램 종료 함수의 일반적 종료 절차 exit

1. 모든 파일 기술자를 닫는다.
2. 부모 프로세스에 종료 상태를 알린다.
3. 자식 프로세스들에 SIGHUP 시그널을 보낸다.
4. 부모 프로세스에 SIGCHLD 시그널을 보낸다.
5. 프로세스간 통신에 사용한 자원을 반납한다.

[예제 6-3] exit, atexit 함수 사용하기

ex6_3.c

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  void cleanup1(void) {
05      printf("Cleanup 1 is called.\n");
06  }
07
08  void cleanup2(void) {
09      printf("Cleanup 2 is called.\n");
10  }
11
12  int main(void) {
13      atexit(cleanup1);
14      atexit(cleanup2);
15
16      exit(0);
17  }
```

종료시 수행할 함수 지정
지정한 순서의 역순으로 실행
(실행결과 확인)

```
# ex6_3.out
Cleanup 2 is called.
Cleanup 1 is called.
```

exec 함수군 활용

exec 함수군

- exec로 시작하는 함수들로, 명령이나 실행 파일을 실행할 수 있다.
- exec 함수가 실행되면 프로세스의 메모리 이미지는 실행파일로 바뀐다.

exec 함수군의 형태 6가지

경로, 인자, 끝알림 순인데 첫인자는 그냥 실행파일명을 쓰는게 일반적으로

```
#include <unistd.h>
int execl(const char *path, const char *arg0, ..., const char *argn,
(char *)0);
int execv(const char *path, char *const argv[]);
int execle(const char *path, const char *arg0, ..., const char *argn,
(char *)0, char *const envp[]);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ..., const char *argn,
(char *)0);
int execvp(const char *file, char *const argv[]);
```

환경변수

경로(파일)

- path : 명령의 경로 지정
- file : 실행 파일명 지정
- arg#, argv : main 함수에 전달할 인자 지정
- envp : main 함수에 전달할 환경변수 지정
- 함수의 형태에 따라 NULL 값 지정에 주의해야 한다.

l : list 처럼 열거 v : 배열을 사용
e : environment p : path

[예제 6-4] execlp 함수 사용하기

ex6_4.c

```
01  #include <unistd.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      printf("--> Before exec function\n");
07
08      if (execlp("ls", "ls", "-a", (char *)NULL) == -1) {
09          perror("execlp");
10          exit(1);
11      }
12
13      printf("--> After exec function\n");
14
15      return 0;
16  }
```

옵션 더 넣고 싶으면 ," " 추가

인자의 끝을 표시하는 NULL 포인터 OS 팀

첫 인자는 관례적으로 실행파일명 지정

메모리 이미지가 'ls' 명령으로 바뀌어 13행은 실행안됨 ☆ 실패해야 정상임

```
# ex6_4.out
--> Before exec function
.       ex6_1.c      ex6_3.c      ex6_4.out
..      ex6_2.c      ex6_4.c      han.txt
```

--> After...

[예제 6-5] execv 함수 사용하기

ex6_5.c

```
01  #include <unistd.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      char *argv[3];
07
08      printf("Before exec function\n");
09
10      argv[0] = "ls";
11      argv[1] = "-a";
12      argv[2] = NULL;
13      if (execv("/bin/ls", argv) == -1) {
14          perror("execv");
15          exit(1);
16      }
17
18      printf("After exec function\n");
19
20      return 0;
21  }
```

13번째줄에서 경로가 ls 이므로 사실 10번째줄 ls 안써도 됨

첫 인자는 관례적으로 실행파일명 지정

인자의 끝을 표시하는 NULL 포인터

경로로 명령 지정

역시 실행안 됨

```
# ex6_5.out
--> Before exec function
.    ex6_1.c   ex6_3.c   ex6_5.c     han.txt
..   ex6_2.c   ex6_4.c   ex6_5.out
```

[예제 6-6] execve 함수 사용하기

ex6_6.c

cc -o arg.out arg.c
이거 실행하기 전에 반드시 arg.c를 미리 컴파일 해둘 것

```
...
05  int main(void) {
06      char *argv[3];
07      char *envp[2];
08
09      printf("Before exec function\n");
10
11      argv[0] = "arg.out";
12      argv[1] = "100";
13      argv[2] = NULL;
14
15      envp[0] = "MYENV=hanbit";
16      envp[1] = NULL;
17
18      if (execve("./arg.out", argv, envp) == -1) {
19          perror("execve");
20          exit(1);
21      }
22
23      printf("After exec function\n");
24
25      return 0;
26  }
```

환경변수를 위한 배열

실행파일명 지정

오류나면 ./arg.out

인자의 끝을 표시하는 NULL 포인터

환경변수 설정

ex6_6_arg.c를 컴파일하여 생성

```
# ex6_6.out
--> Before exec function
--> In ex6_6_arg.c Main
argc = 2
argv[0] = arg.out
argv[1] = 100
MYENV=hanbit
```

[예제 6-6] (2) ex6_6_arg.c 파일

ex6_6_arg.c

```
01  #include <stdio.h>
02
03  int main(int argc, char **argv, char **envp) {
04      int n;
05      char **env;
06
07      printf("\n--> In ex6_6_arg.c Main\n");
08      printf("argc = %d\n", argc);
09      for (n = 0; n < argc; n++)
10          printf("argv[%d] = %s\n", n, argv[n]);
11
12      env = envp;
13      while (*env) {
14          printf("%s\n", *env);
15          env++;
16      }
17
18      return 0;
19  }
```

인자 값 출력

전부

환경변수 출력

MYENV = hanbit

exec 함수군과 fork 함수

- fork로 생성한 자식 프로세스에서 exec 함수군을 호출
  - 자식 프로세스의 메모리 이미지가 부모 프로세스 이미지에서 exec 함수로 호출한 새로운 명령으로 대체
  - 자식 프로세스는 부모 프로세스와 다른 프로그램 실행 가능
  - 부모 프로세스와 자식 프로세스가 각기 다른 작업을 수행해야 할 때 fork와 exec 함수를 함께 사용

[예제 6-7] fork와 exec 함수 사용하기

ex6_7.c

```
...
06  int main(void) {
07      pid_t pid;
08
09      switch (pid = fork()) {
10          case -1 :  /* fork failed */
11              perror("fork");
12              exit(1);
13              break;
14          case 0 :   /* child process */
15              printf("--> Child Process\n");
16              if (execlp("ls", "ls", "-a", (char *)NULL) == -1) {
17                  perror("execlp");
18                  exit(1);
19      }
20              exit(0);
21              break;
22          default :   /* parent process */
23              printf("--> Parent process - My PID:%d\n",(int)getpid());
24              break;
25  }
27      return 0;
28   }
```

자식프로세스에서 execlp 함수 실행

부모프로세스는 이 부분 실행

```
# ex6_7.out
--> Child Process
ex6_1.c  ex6_3.c  ex6_5.c  ex6_6_arg.c  ex6_7.out
ex6_2.c  ex6_4.c  ex6_6.c  ex6_7.c      han.txt
--> Parent process - My PID:10535
```

프로세스 동기화

부모 프로세스와 자식 프로세스의 종료절차

동기화: 부모프로세스는 자식이 죽을때까지 기다림

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

좀비프로세스

- 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

고아프로세스

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

프로세스 동기화 함수[1]

프로세스 동기화: wait(3)

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

- stat_loc : 상태정보를 저장할 주소
- 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

참고로 여기선 sleep(3) 이면 3초임

[예제 6-8] wait 함수 사용하기

ex6_8.c

```
...
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  }
```

Status: 0x256, 0+"00"

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

Status는 exit(2)에서 오는 2를 받기 위한 변수

자식 프로세스의 종료를 기다림

오른쪽으로 8비트 이동해야 종료 상태값을 알 수 있음 (x256)

2를 받았지만 2가 그대로 출력되는건 X

프로세스 동기화 함수[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

[예제 6-9] waitpid 함수 사용하기

ex6_9.c

```
...
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

Thank You !

IT CookBook, 유닉스 시스템 프로그래밍

한빛미디어 Hanbit Media, Inc.