Operating Systems LAB 7

Wonpyo Kim
skykwp@gmail.com

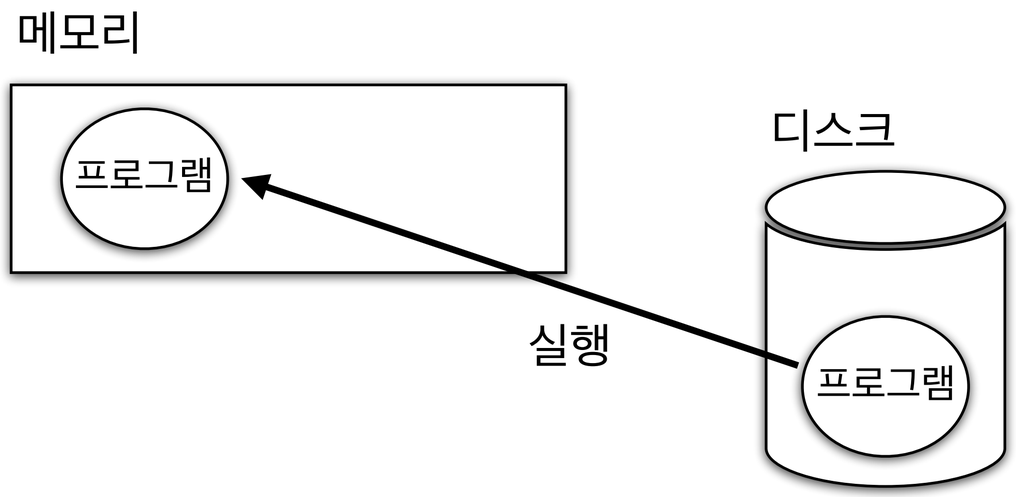
Outline

- Substance
  - **Process Management**

Process Management

- 프로그램이 실행되기 위해서는 메모리에 적재되어야 한다. 즉, 메모리에 적재되어 실행 중에 있는 프로그램을 프로세스(process)라 한다.

- 프로세스는 스케쥴링의 대상이 되는 프로그램으로, 모든 프로세스에는 유일한 번호가 배정되는데, 이를 프로세스 ID(process ID)라 한다.



Process Management - fork( )

- 리눅스에서 새로운 프로세스를 생성하는 방법은 이미 존재하는 프로세스가 fork( ) 함수를 호출하는 것이다.
- fork( ) 함수는 현재의 프로세스를 복제하여 동일한 프로세스를 생성한다.

fork 함수
기능
    자식 프로세스를 생성한다.
기본형
    pid_t fork(void);
반환값
    성공:
        부모 프로세스: 자식 프로세스의 프로세스 ID
        자식 프로세스: 0
    실패: -1
헤더파일
    <sys/types.h>
    <unistd.h>

Process Management - fork( )

- fork( ) 에 의해 생성되는 자식 프로세스는 부모 프로세스와 코드가 동일하다. 이 두 프로세스를 구별하기 위해서는 fork( ) 에 의해 반환되는 값을 확인하는 것이다.
- 부모 프로세스에게 반환되는 값은 자식 프로세스의 ID가 되고, 자식 프로세스에서는 0이 반환되므로 구분할 수 있다.
- 부모 프로세스가 가질 수 있는 자식 프로세스의 수에 제한이 있으므로 fork( ) 호출에 실패할 수 있는데, 이 때는 -1 이 반환된다.

Process Management - fork( )

- 예제

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

main()
{
    pid_t pid;

    if ((pid = fork()) == -1) {
        perror("fork failed");
    } else if (pid != 0) {
        printf("parent process\n");
    } else {
        printf("child process\n");
    }

}
```

부모 프로세스

```c
} else if (pid != 0) {
    printf("parent process\n");
} else {
    printf("child process\n");
}
```

자식 프로세스

```c
} else if (pid != 0) {
    printf("parent process\n");
} else {
    printf("child process\n");
}
```

- 'pid != 0' 이 참일 때, 부모 프로세스는 자식의 pid를 갖기 때문에 부모 프로세스의 조건이다.
- 그 이외의 경우인 else 는 0 이므로 자식 프로세스의 동작을 의미한다.

Process Management - getpid( ), getppid( )

- 문제는 PID 는 항상 가변적이기 때문에 효율적인 동작을 위해서는 PID 를 찾는 함수를 사용해야 한다.
- getpid( ), getppid( ) 함수는 각각 자식과 부모의 PID 를 반환해준다.

```
getpid, getppid 함수
기능
        getpid는 자신의 프로세스 ID, getppid는 부모의 프로세스 ID를 반환한다.
기본형
        pid_t getpid(void);
        pid_t getppid(void);
반환값
        성공: 프로세스 ID
        실패: 발생하지 않음
헤더파일
        <unistd.h>
```

Process Management - getpid( ), getppid( )

- 예제

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

main()
{
    pid_t pid;
    if ((pid = fork()) == -1) {
        perror("fork failed");
    } else if (pid != 0) {
        printf("PID is %d, child process pid is %d\n", getpid(), pid);
    } else {
        printf("PID is %d, parent process pid is %d\n", getpid(), getppid());
    }
}
```

Process Management - vfork( )

- vfork( ) 는 fork( ) 와 같이 새로운 프로세스를 생성하며 반환되는 값도 동일하다.
- 차이점은 자식 프로세스가 exit( ) 나 exec( ) 를 호출할 때까지 부모 프로세스는 실행되지 않고 기다린다는 점이다.

```
vfork 함수
기능
        자식 프로세스를 생성하고 부모 프로세스는 기다린다.
기본형
        pid_t vfork(void);
반환값
        성공:
            부모 프로세스: 자식 프로세스의 프로세스 ID
            자식 프로세스: 0
        실패: -1
헤더파일
        <sys/types.h>
        <unistd.h>
```

Process Management - vfork( )

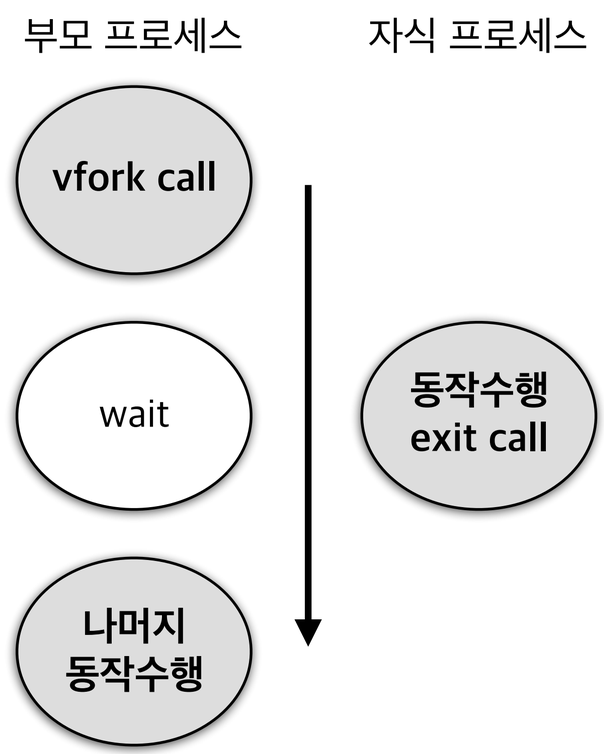
- 예제

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

main()
{
    pid_t pid;

    if ((pid = vfork()) == -1) {
        perror("fork failed");
    }

    if (pid != 0) {
        printf("PID is %d, child process pid is %d\n", getpid(), pid);
    } else {
        printf("PID is %d, parent process pid is %d\n", getpid(), getppid());
        sleep(5);
        exit(0);
    }
}
```



Process Management - exit( )

- exit( ) 함수를 호출하면 프로세스가 "정상적"으로 종료된다.
- 아래의 status 의 하위 8비트는 main( ) 함수의 반환값이 된다. 이 반환값을 보고 종료 상태를 알 수 있게 된다.

```
exit 함수
기능
        정상적으로 프로세스를 종료한다.
기본형
        void exit(int status);
        status: main 함수의 반환값
반환값
        없음
헤더파일
        <stdlib.h>
```

Process Management - atexit( )

- atexit( ) 함수는 exit( ) 함수가 호출되거나 main( ) 함수가 리턴할 때 실행될 함수를 등록하는 함수이다.

- exit( ) 함수를 호출하면, atexit( ) 에 의해 등록된 함수가 실행된다. 만약 등록된 함수가 여러 개일 경우 최근에 등록된 함수순으로 실행되고 인수에 함수 이름이 위치한다.

```
atexit 함수
기능
        exit 호출 때 실행할 함수를 등록한다.
기본형
        int atexit(void (*function)(void));
        function: 등록할 함수 이름으로 인수가 없는 함수
반환값
        성공: 0
        실패: -1
헤더파일
        <stdlib.h>
```

Process Management - atexit( )

- 예제

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

void func1(void);
void func2(void);
void func3(void);

main()
{
    atexit(func1);
    atexit(func2);
    atexit(func3);

    exit(0);
}

void func1(void)
{
    printf("I'm function 1\n");
}

void func2(void)
{
    printf("I'm function 2\n");
}

void func3(void)
{
    printf("I'm function 3\n");
}
```

- 본 예제는 최근에 등록된 함수 순인 func3( ) -> func2( ) -> func1( ) 의 순서로 main( ) 함수가 exit( ) 함수의 호출로 종료될 때, 순차적으로 실행된다.

Process Management - atexit( )

- abort( ) 함수는 현재 상태를 코어 덤프하고 프로세스를 비정상 종료한다.
- 이처럼 프로세스는 정상 종료와 비정상 종료 두 가지로 나뉘게 되며, 비정상 종료일 때는 atexit( ) 에 의해 등록된 함수가 수행되지 않는다.

정상적인 종료	비정상적인 종료
exit 에 의한 종료	abort 에 의한 종료
return 에 의한 종료	시그널에 의한 종료

Process Management - atexit( )

- 예제

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

void func(void);

main(int argc, char *argv[])
{
    atexit(func);
    if (!atoi(argv[1])) {
        abort();
    }

    exit(0);
}

void func(void)
{
    printf("I'm here.\n");
}
```

- 본 예제는 argv 를 사용하여 1이 입력될 때, 정상 종료되고 0이 입력될 때는 abort( ) 함수에 의해 코어 덤프가 된다.
- 실행하면 1일 때만 func( ) 함수가 호출되고 그 외에는 코어 덤프가 발생한다.

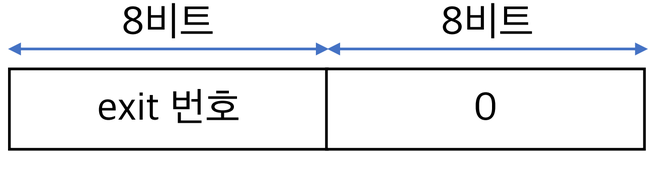
Process Management - wait( )

- wait( ) 함수는 자식 프로세스가 종료될 때까지 아무 일도 하지 않고 기다린다.

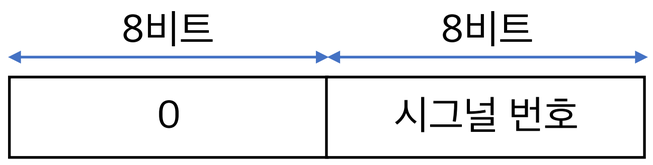
wait 함수
기능
　　자식 프로세스가 종료될 때까지 기다린다.
기본형
　　pid_t wait(int *status);
　　status: 자식 프로세스가 종료될 때의 상태 정보
반환값
　　성공: 종료된 자식 프로세스의 프로세스 ID
　　실패: -1
헤더파일
　　<sys/types.h>
　　<sys/wait.h>

Process Management - wait( )

- status 에는 자식 프로세스가 종료될 때의 상태 정보가 저장되는데, 자식 프로세스가 정상적으로 종료되면 status 의 하위 8비트에는 0이 저장되고, 상위 8비트에는 exit(번호)의 "번호" 가 저장된다.



- 그러나, 비정상적으로 종료되면 status 의 우측 8비트에 프로세스를 종료시킨 시그널의 번호가 저장된다.



Process Management - wait( )

- 예제

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

main()
{
    int pid, child_pid, status;

    if ((pid = fork()) == -1) {
        perror("fork falied");
    }

    if (pid != 0) {
        child_pid = wait(&status);
        printf("child[PID:%d] terminated with code %x\n", child_pid, status);
    } else {
        printf("run child...\n");
        exit(3);
    }
}
```

- 본 예제의 출력결과는 300 이라는 코드를 동반한다. %x 형태로 출력했기 때문에 16비트로 출력된다.

Process Management - waitpid( )

- wait( ) 과 유사한 함수로 waitpid( ) 가 있다. 차이점은 wait( ) 함수의 경우 자식 프로세스 중 "하나라도" 종료되면 부모 프로세스는 깨어나지만, waitpid( ) 함수는 특정 프로세스가 종료되기를 기다리도록 지정할 수 있다.

- wait( ) 함수는 부모 프로세스가 아무 일도 하지 않고 블록화되지만, waitpid( ) 함수에서는 자식 프로세스가 종료되지 않아도 부모 프로세스가 블록화 되지않고 다른 일을 하도록 지정할 수 있다.

```
waitpid 함수
기능
        (특정) 자식 프로세스가 종료될 때까지 기다린다.
기본형
        pid_t waitpid(pid_t pid, int *status, int options);
        pid: 종료를 기다리는 자식 프로세스의 프로세스 ID
        status: 자식 프로세스가 종료될 때의 상태 정보
        options: 옵션
반환값
        성공: 종료된 자식 프로세스의 프로세스 ID
        실패: -1
헤더파일
        <sys/types.h>
        <sys/wait.h>
```

Process Management - waitpid( )

- waitpid( ) 함수의 첫 번째 매개변수인 pid 는 다음과 같은 의미를 갖는다.

pid	의미
-1	여러 자식 프로세스 중 하나라도 종료되기를 기다린다.
0	호출한 프로세스의 그룹 ID와 같은 그룹 ID를 가지는 자식 프로세스가 종료되기를 기다린다.
양수	pid번 프로세스가 종료되기를 기다린다.

- options 매개변수는 <sys/wait.h> 에 정의된 여러 값을 취할 수 있으나, 가장 유용한 것은 WNOHANG 으로 이를 설정하면 자식 프로세스가 종료되지 않더라도 부모 프로세스는 블록화되지 않고 다른 일을 수행한다.

- options 매개변수에 0을 설정하면 wait 과 동일하게 자식 프로세스가 종료될 때까지 부모 프로세스는 블록화된다.

options	의미
WNOHANG	자식 프로세스가 종료되지 않더라도 부모 프로세스는 블록화되지 않고 다른 일을 수행한다.
0	wait과 동일하게 자식 프로세스가 종료될 때까지 부모 프로세스는 블록화된다.

Process Management - waitpid( )

- 예제

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

main()
{
    int pid, child_pid, status;

    if ((pid = fork()) == -1) {
        perror("fork failed");
    }

    if (pid != 0) {
        if ((pid = fork()) == -1) {
            perror("fork failed");
        }
        if (pid != 0) {
            child_pid = wait(&status);
            printf("child[PID:%d] terminated with code %x\n", child_pid, status);
        } else {
            printf("run child2[PID:%d]\n", getpid());
            sleep(5);
            printf("run child2 finished\n");
            exit(3);
        }
    } else {
        printf("run child1[PID:%d]\n", getpid());
        exit(2);
    }
}
```

- 본 예제에서는 부모 프로세스가 fork( ) 를 수행한 후, 다시 fork( ) 를 수행해서 자식을 2개 생성한다.
- 추가로 생성되는 child2 의 부모 프로세스는 wait( ) 함수를 통해 기다리고 있지만, exit(2) 를 수행하는 자식 프로세스 1에 의해 먼저 종료가 된다.
- 하지만, 자식 2는 sleep(5) 에 의해 5초 후에 추가적인 출력문을 출력하고 종료한다.

Process Management - waitpid( )

- 예제

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

main()
{
    int pid1, pid2, child_pid, status;

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

    if (pid1 != 0) {
        if ((pid2 = fork()) == -1) {
            perror("fork failed");
        }
        if (pid2 != 0) {
            //child_pid = wait(&status);
            child_pid = waitpid(pid2, &status, 0);
            printf("child[PID:%d] terminated with code %x\n", child_pid, status);
        } else {
            printf("run child2[PID:%d]\n", getpid());
            sleep(5);
            printf("run child2 finished\n");
            exit(3);
        }
    } else {
        printf("run child1[PID:%d]\n", getpid());
        exit(2);
    }
}
```

- 본 예제는 이전 페이지와 동일한 코드이지만, wait( ) 함수를 waitpid( ) 함수로 바꾸고 자식 2의 프로세스를 기다리도록 하였다.
- 이전 예제에서는 부모가 먼저 종료되고 5초 후에 자식의 출력문이 나타났지만, 본 예제에서는 5초 후 자식의 출력문을 부모가 기다리게 된다.
- 반면에, waitpid( ) 의 options 인자인 0을 WNOHANG으로 바꾸면, 부모는 블록화되지 않기 때문에 이전 예제와 같은 출력을 보이게 된다.

Process Management - waitpid( )

- 종료 상태 정보 매크로
  - wait( ), waitpid( ) 함수를 이용할 때, 다음과 같은 매크로를 사용하면 종료할 때의 상태 정보를 얻을 수 있다.

매크로	의미
WIFEXITED(status)	자식 프로세스가 정상적으로 종료되면 참이 반환된다.
WIFSIGNALED(status)	자식 프로세스가 시그널에 의해 비정상적으로 종료되면 참이 반환된다.
WIFSTOPPED(status	자식 프로세스가 중단되었다면 참이 반환된다.
WEXITSTATUS(status)	자식 프로세스가 정상적으로 종료되면 자식 프로세스의 종료 코드를 반환한다.

Process Management - waitpid( )

- 예제

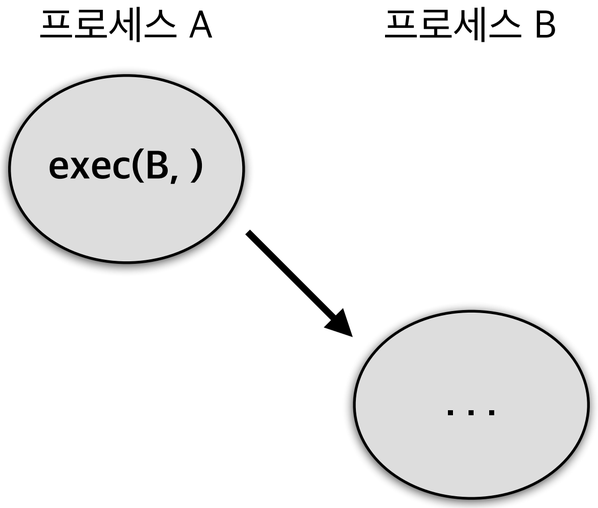
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

main()
{
    int pid, child_pid, status;
    if ((pid = fork()) == -1) {
        perror("fork failed");
    }

    if (pid != 0) {
        pid = wait(&status);
        if (WIFEXITED(status)) {
            printf("child terminated with code %d\n", WEXITSTATUS(status));
        } else {
            printf("child terminated abnormally\n");
        }
    } else {
        printf("run child\n");
        exit(27);
    }
}
```

Process Management - exec( )

- exec( ) 함수는 하나의 프로세스가 실행되다가 또 다른 프로세스를 생성하여 실행시킨 후 자기 자신은 종료하는 함수이다.
- 다음은 프로세스 A가 실행되다가 exec( ) 함수를 호출하면 프로세스 B로 대체된다.



Process Management - exec( )

- exec( ) 함수는 인수를 표현하는 방법에 따라 6가지가 존재한다. 크기 execl( ) 계열과 execv( ) 계열이 있다.

execl, execlp, execle, execv, execvp, execve 함수
기능
　　다른 프로세스를 실행시킨다.
기본형
　　int execl(const char *path, const char *arg, ..., (char *)0);
　　int execlp(const char *file, const char *arg, ..., (char *)0);
　　int execle(const char *path, const char *arg, ..., (char *)0, const char *envp[]);
　　int execv(const char *path, const char *argv[]);
　　int execvp(const char *file, const char *argv[]);
　　int execve(const char *path, const char *argv[], const char *envp[]);
　　path: 실행될 파일의 경로이름
　　file: 실행될 파일의 이름
　　arg: 실행 파일의 인수들
　　argv: 실행 파일의 인수 배열
　　envp: 환경 변수
반환값
　　성공: 없음
　　실패: -1
헤더파일
　　<unistd.h>

Process Management - exec( )

- execl( ), execlp( ), execle( )의 execl( ) 계열은 NULL(char *)0 으로 끝나는 리스트가 인수가 된다.
- execv( ), execvp( ), execve( )의 execv( ) 계열은 문자열 배열이 두 번째 인수가 된다. 여기에서 (char *)0 은 인수가 더 이상 없음을 의미한다.
- 인수 중 path 는 '/bin/ls' 와 같이 새롭게 실행될 프로그램의 경로를 의미한다.
- 반면에, file 이라고 명시된 p 함수들은 경로를 나타내지 않고, 프로그램 이름만을 나타낸다. 이 경우, PATH 환경변수에 의해 지정된 디렉토리에서 프로그램을 찾는다.
- envp 인수의 경우 새로운 프로그램의 환경 정보를 임의로 설정하고자 할 때 사용한다.

Process Management - execl( ), execlp( )

- 예제

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
    printf("running with execl\n");

    execl("/bin/ls", "ls", "-al", (char *)0);

    printf("execl failed to run ls\n");

    exit(0);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
    printf("running with execl\n");

    execlp("ls", "ls", "-al", (char *)0);

    printf("execl failed to run ls\n");

    exit(0);
}
```

- execl 의 첫 번째 인수는 ls 파일의 위치, 두 번째와 세 번째는 ls 의 인수를 주고자 할 때 사용한다.
- 오른쪽의 예제는 PATH 환경변수를 참조하여 첫 번째 인수를 실행한다.

Process Management - execle( )

- 예제

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

char *envp[] = {"USER=s_guest", "PATH=/tmp", (char *)0};

main()
{
    printf("running ls with execle\n");

    execle("/bin/ls", "ls", "-al", (char *)0, envp);

    printf("execle failed to run ls\n");
    exit(0);
}
```

Process Management - execv( ), execvp( )

- 예제

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
    char *arg[] = {"ls", "-l", (char *)0};

    printf("running ls with execv\n");

    execv("/bin/ls", arg);

    printf("execv failed to run ls\n");
    exit(0);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
    char *arg[] = {"ls", "-l", (char *)0};

    printf("running ls with execvp\n");

    execvp("ls", arg);

    printf("execvp failed to run ls\n");
    exit(0);
}
```

Process Management - execvp( )

- 응용 예제

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main(int argc, char *argv[])
{
    int i;
    char *arg[argc];

    printf("running %s with execvp\n", argv[1]);

    for (i = 0; i < (argc-1); i++) {
        arg[i] = argv[i+1];
    }

    arg[i] = (char *)0;
    execvp(argv[1], arg);

    printf("execvp failed to run %s\n", argv[1]);
    exit(0);
}
```

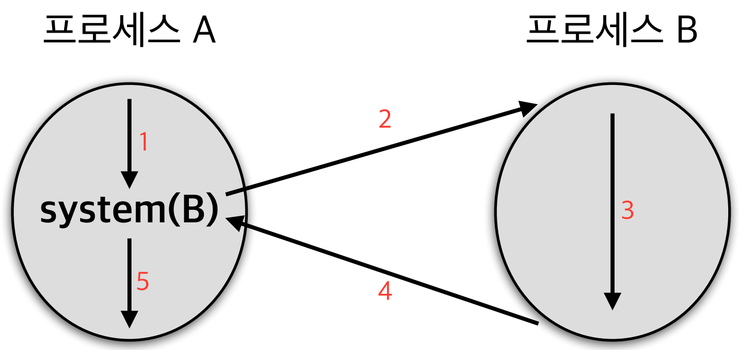
Process Management - system( )

- system( ) 함수를 사용하면 다른 프로그램이 실행되도록 할 수 있다. exec( ) 함수와의 차이점은 system( ) 함수는 다른 프로세스의 실행이 종료될 때 까지 기다려준다.

```
system 함수
기능
        새로운 프로세스를 실행한다.
기본형
        int system(const char *string);
        string: 실행할 명령어 경로
반환값
        성공: -1, 127 외의 값
        실패: -1 또는 127
헤더파일
        <stdlib.h>
```

Process Management - system( )

- 프로세스 A에서 system(B) 를 호출하면 프로세스 B가 실행되고 실행이 끝나면 프로세스 A의 system( ) 함수 호출 명령어 다음의 내용을 실행하게 된다.



- system( ) 함수 내부적으로는 fork( ), exec( ), wait( ) 로 구현되어 있다.
- system( ) 함수가 -1과 127 이외의 값을 반환하면 호출에 성공한 것이다. -1은 특히 fork( ) 호출이 실패한 것이고, 127은 exec( ) 호출에서 실패한 것이다.

Process Management - system( )

- 예제

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main(int argc, char *argv[])
{
    char str[32] = "";
    int i = 1;

    printf("running %s with system\n", argv[1]);
    while (argc > i) {
        strcat(str, argv[i++]);
        strcat(str, " ");
    }

    system(str);
    printf("Done\n");
    exit(0);
}
```