System Programming

Exercise

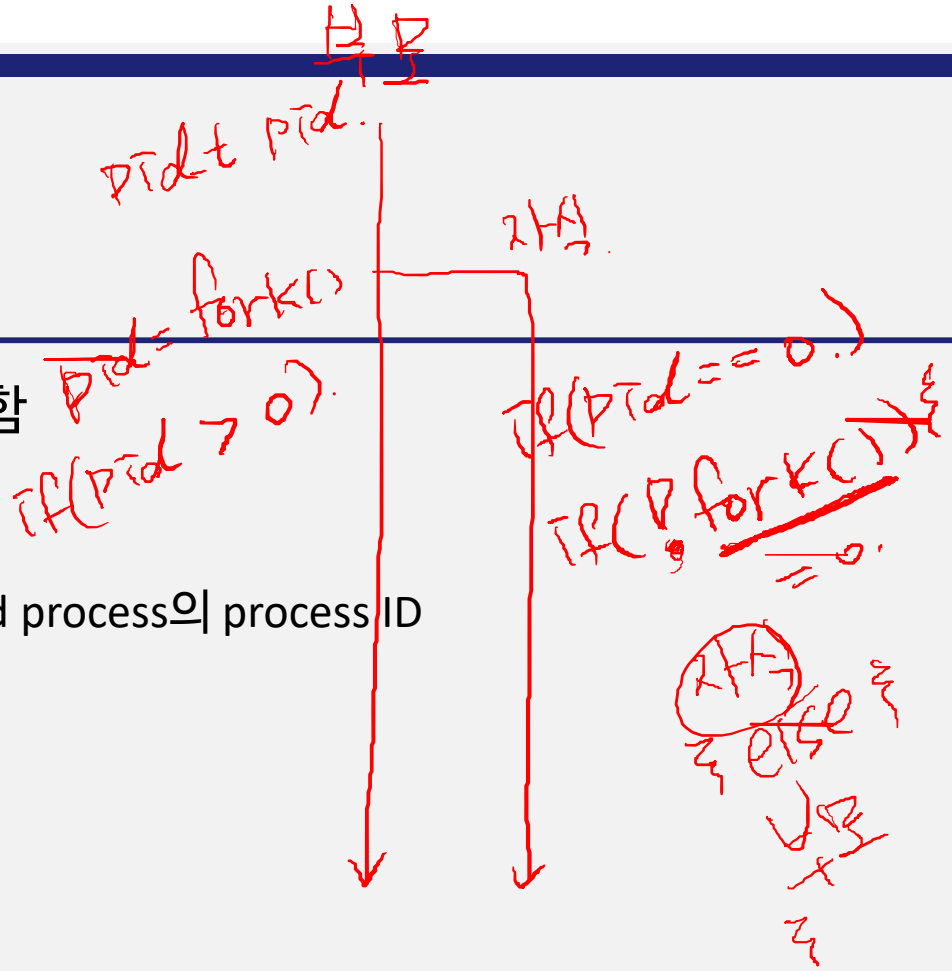
Week 05. Process

Process의 개념

- 수행 중인 프로그램 (Program in execution)
- 각 프로세스는 text, data, stack 영역을 메모리에 할당 받음
- 프로세스(child)는 다른 프로세스(parent)에 의해 생성됨
- 두 프로세스는 프로세스간 통신 (IPC, Inter-Process Communication) 기능을 이용하여 정보를 주고 받을 수 있음
  - UNIX IPCs: signal, pipe, fifo, socket, …
- 프로세스 관련 시스템 호출
  - fork: 자신의 프로세스를 복제하여 child 프로세스를 생성
  - exec: 자신의 프로세스에 다른 프로그램을 덮어 씀
  - wait: child 프로세스가 종료할 때까지 기다림
  - exit: 자신의 프로세스를 종료하며, 상태 정보 반환

System Call – fork()

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

pid_t fork(void);
```



- 자식 프로세스를 생성함
- return value
  - parent process: child process의 process ID
  - child process: 0
  - 실패: -1

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – fork #1

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

int main(int argc, char **argv) {

  int pid;
  if ((pid = fork()) > 0) {
    // getpid() 현재 프로세스의 프로세스ID를 되돌려 줌
    printf("부모 프로세스 %d : %d\n", getpid(), pid);
    return 0;
  } else if (pid == 0) {
    printf("자식 프로세스 %d\n", getpid());
    return 0;
  } else {
    perror("fork error : ");
    exit(0);
  }
  return 0;
}
```

```
./main
부모 프로세스 209 : 210
자식 프로세스 210
```

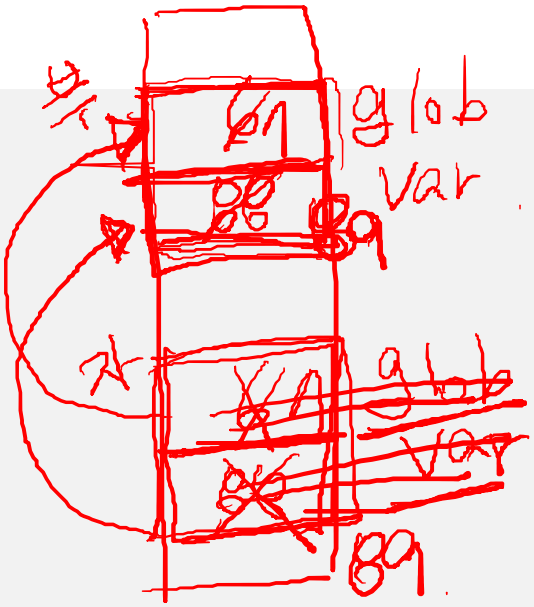
- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – fork #2 (1/2)

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

int glob = 6;
char buf[] = "a write to stdout\n";
int main(void) {
  int var, size;
  pid_t pid;

  var = 88;
  size = sizeof(buf)-1;

  if (write(STDOUT_FILENO, buf, size) != size) {
    perror("write error");
    exit(1);
  }
  /* we don't flush stdout */
  printf("before fork\n");
```

Example – fork #2 (2/2)

```
  if ((pid = fork()) < 0) {
    perror("fork error");
    exit(1);
  } else if (pid == 0) { /* child */
    glob++; /* modify variables */
    var++;
  } else {
    sleep(2); /* parent */
  }
  printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
  exit(0);
}
```




```
a write to stdout
before fork
pid = 402, glob = 7, var = 89
pid = 401, glob = 6, var = 88
```

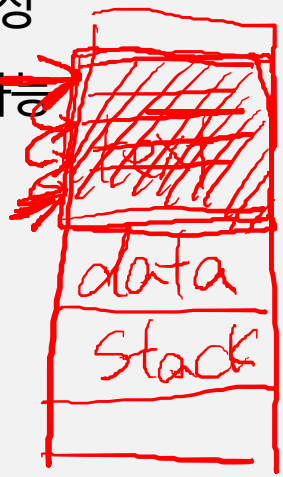
with vfork()

```
pid = 409, glob = 7, var = 89
pid = 408, glob = 7, var = 89
```

System Call – exec()

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

- 현재 프로세스 이미지를 새로운 프로세스 이미지로 바꿈
  - 새 프로그램의 수행이 시작되어, 새 data와 새 stack 형성
  - 이전에 open 된 파일 descriptor는 exec 후에도 사용 가능
  - 이미 open된 파일 descriptor를 close할 수도 있음



System Call – exec()

- exec 동작 상세

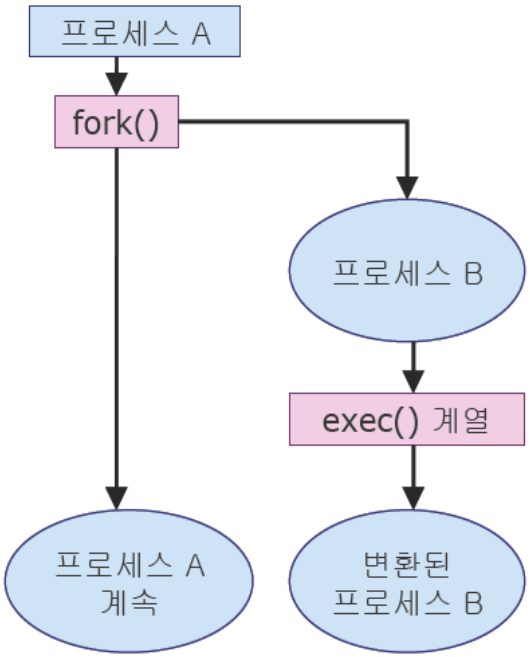
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
  printf("executing ls\n");

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

  perror("execl failed to run ls");
  exit(1);
}
```

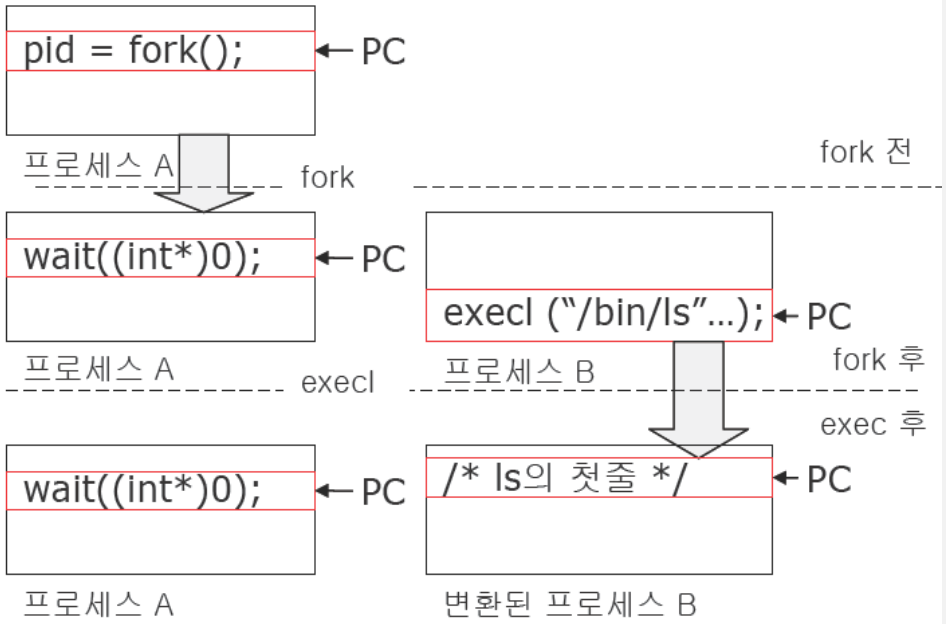
execle
execve

```c
// PATH 환경변수에 /bin이 있다고 가정하면
// execl()함수와 아래 함수는 같은 의미를 갖게 됨
exelp("ls", "ls", "-l", (char *) 0);
char *const vector[] = {"ls", "-l", (char *) 0}
execv("/bin/ls", vector);
ecevp("ls", vector);
```

System Call – exec()

- exec 동작 상세





- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec #1

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

int main() {
  printf("원래 프로세스: %d\n", getpid());
  sleep(1);
  execl("/bin/sh", "sh", NULL);
  exit(0);
}
```

```
원래 프로세스: 38
$ ps
  PID TTY          TIME CMD
   38 pts/0    00:00:00 sh
   40 pts/0    00:00:00 ps
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec #2 (1/3)

```
// echoall.c 파일을 만들고 아래 내용 작성
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int i;
  char **ptr;
  extern char **environ;
  for(i = 0; i < argc; i++)
    printf("argv[%d]: %s\n", i, argv[i]);
  for(ptr = environ; *ptr != 0; ptr++)
    printf("%s\n", *ptr);
  exit(0);
}
```

Example – exec #2 (2/3)

```c
// main.c 파일에 아래 내용 작성
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void) {
  pid_t pid;
  char path[3000];
  char *vector[] = {"echoall", "myarg1", "MY AGR2", (char *) 0};
  if ((pid = fork()) < 0) {
    perror("fork error");
    exit(1);
  } else if (pid == 0) {
    /* specify pathname, specify environment */
    sprintf(path, "PATH=%s", getcwd(NULL, 0));
    char *env_init[] = {"USER=unknown", path, NULL};
    if (execve("echoall", vector, env_init) < 0) {
     fprintf(stderr, "execve error\n");
     exit(1);
    }
  }
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec #2 (3/3)

```
  if (waitpid(pid, NULL, 0) < 0) {
    perror("waitpid error");
    exit(1);
  }
  if ((pid = fork()) < 0) {
    perror("fork2 error");
    exit(1);
  } else if (pid == 0) {
    if (execlp("echoall", "echoall", "only one arg", (char*) 0) < 0) {
      perror("execle error");
      exit(1);
    }
  }
  return 0;
}
```

```
user@user-VirtualBox:~/eclipse-workspace/Lab5-4-exec2/src$ ./m
ain
argv[0]: echoall
argv[1]: myarg1
argv[2]: MY ARG2
USER=unknown
PATH=/home/user/eclipse-workspace/Lab5-4-exec2/src
user@user-VirtualBox:~/eclipse-workspace/Lab5-4-exec2/src$ exe
clp error: No such file or directory
```

System Call – exit

```
#include <stdlib.h>

void exit(int status);
```

- 프로그램을 정상 종료 시킴
- status
  - 성공 시: 0
  - 실패 시: non-zero
- no return value

System Call – wait

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

- 자식 프로세스가 종료될 때까지
  해당 영역에서 부모 프로세스가 sleep 모드로 기다림
  - 부모 프로세스가 자식 프로세스보다 먼저 종료되어
    자식 프로세스를 고아로 만들지 않기 위함
- status
  - 자식 프로세스의 종료 상태 값을 받아 옴
- return value
  - 성공 시: 종료한 자식 프로세스의 ID 값
  - 실패 시: -1

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – wait (1/2)

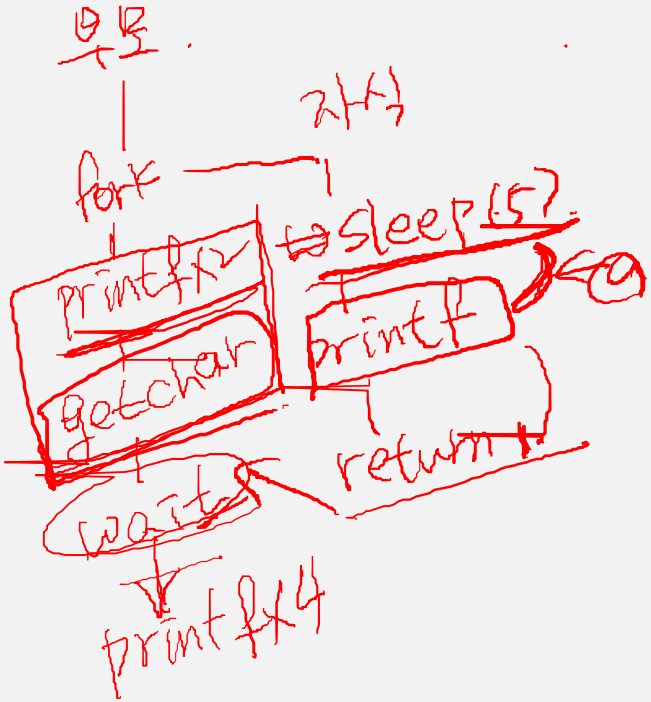
```c
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
int main() {
  int pid;
  int status;
  int spid;
  if ((pid = fork()) == 0) {
    sleep(5);
    printf("I will be back %d\n", getpid());
    return 1;
  } else if(pid > 0) {
    printf("I'm parent %d\n", getpid());
    printf("Press any key and wait\n");
    getchar();
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – wait (2/2)

```
    // 자식프로세스를 wait 한다.
    // 자식프로세스의 종료 상태는 status를 통해 받아온다.
    spid = wait(&status);
    printf("자식프로세스 wait 성공\n");
    // 자식프로세스의PID, 리턴 값, 종료 상태(정상 혹은 비정상종 료)를 얻어옴
    printf("PID: %d\n", spid);
    printf("Exit Value : %d\n", WIFEXITED(status));
    printf("Exit Stat : %d\n", WEXITSTATUS(status));
  } else {
    perror("fork error :");
  }
}
```

```
I'm parent 113
Press any key and wait
a
I will be back 114
자식프로세스 wait 성공
PID: 114
Exit Value : 1
Exit Stat : 1
```



System Call – waitpid (1/2)

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pit_t pid, int *status, int option);
```

- 주어진 pid의 자식 프로세스가 종료되길 기다림
  - pid == -1: 임의의 자식 프로세스를 기다림 (=wait())
  - pid == 0: 프로세스 그룹ID가 호출 프로세스의 ID와 동일한 자식 프로세스를 기다림
  - pid < 0: 그룹ID가 pid의 절댓값과 같은 자식 프로세스를 기다림
  - pid > 0: 프로세스ID가 pid 값과 같은 자식 프로세스를 기다림
- options
  - WNOHANG: 자식을 기다리지 않고 다른 작업 수행
  - WCONTINUED: 자식이 재개된 경우에 보고
  - WUNTRACED: 자식이 중지된 경우 보고
- 성공 시: 종료한 자식 프로세스의 ID 값, 실패 시: -1

System Call – waitpid (2/2)

- 매크로 함수
  - status: 프로세스의 상태정보를 가져오기 위해 사용
  - status가 NULL이 아닌 경우, 상태정보를 저장함
  - 아래 매크로를 이용하여 상태정보를 가져올 수 있음

매크로	설 명
WIFEXITED(status)	자식 프로세스가 정상적으로 종료된 경우 true 반환한다.
WEXITSTATUS(status)	WIFEXITED가 true인 경우, 종료된 자식 반환 코드의 최하위 8비트의 값을 반환한다.
WIFSIGNALED(status)	자식 프로세스가 시그널을 받아서 비정상 적으로 종료된 경우 true를 반환한다.
WTERMSIG(status)	WIFSIGNALED가 true를 반환한 경우, 자식 프로세스를 종료시킨 시그널의 번호를 반환한다.
WIFSTOPPED(status)	자식 프로세스의 수행이 중지된 경우 true를 반환한다. (WUNTRACED옵션이 설정된 경우)
WSTOPSIG(status)	WIFSTOPPED가 true를 반환한 경우, 자식 프로세스를 중지시킨 시그널의 번호를 반환한다.
WIFCONTINUED(status)	프로세스의 수행이 재개된 경우 true를 반환한다. (WCONTINUED 옵션이 설정된 경우)
WCOREDUMP(status)	core dump라고 불리는 메모리 덤프 파일이 생성된 경우에만 true를 반환한다. (프로세스가 죽은 원인을 분석하는데 유용)

Example – waitpid (1/2)

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

int main() {
  int pid;
  int status;
  if ((pid = fork()) < 0) {
   perror("fork error : ");
   exit(0);
  } else if (pid == 0) {
   printf("I'm Child\n");
   sleep(10);
   return 2;
  }
```

Example – waitpid (2/2)

```
    else {
      printf("Parent: wait (%d)\n", pid);
      waitpid(pid, &status, 0);
      if (WIFEXITED(status)){
       printf("정상 종료\n");
       printf("리턴 값 %d\n", WEXITSTATUS(status));
    } else if (WIFSIGNALED(status)){
      printf("신호 받았음\n");
      printf("신호 번호 %d\n", WTERMSIG(status));
    }
  }
  exit(0);
}
```

```
Parent: wait (159)
I'm Child
정상 종료
리턴 값 2
```

```
./main&
[1] 52
Parent: wait (53)
I'm Child
kill -KILL 53
신호 받았음
신호 번호 9
^C
[1]+  Done
      ./main
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Process ID 관련 System Calls

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);          // 현재 프로세스의 프로세스ID를 되돌림
pid_t getppid(void);         // 부모 프로세스의 PID를 되돌림
gid_t getgid(void);          // 현재 프로세스의 실제 프로세스그룹ID를 되돌림
gid_t getegid(void);         // 현재 프로세스의 유효 프로세스그룹ID를 되돌림
uid_t getuid(void);          // 현재 프로세스의 실제 프로세스사용자ID를 되돌림
uid_t geteuid(void);         // 현재 프로세스의 유효 프로세스사용자ID를 되돌림
pid_t getsid(pid_t pid);     // pid를 갖는 프로세스의 세션ID를 되돌림
int setgid(gid_t gid);       // 실제그룹ID를 변경함
int setegid(gid_t gid);      // 유효그룹ID를 변경함
int setuid(uid_t uid);       // 실제사용자ID를 변경함
int seteuid(uid_t uid);      // 유효사용자ID를 변경함
pid_t setsid(void);          // 세션을 만들고 프로세스그룹아이디(GID)를 설정
                    // 만약 호출프로세스가 프로세스 리더이면 함수는 실패하고 -1 반환
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

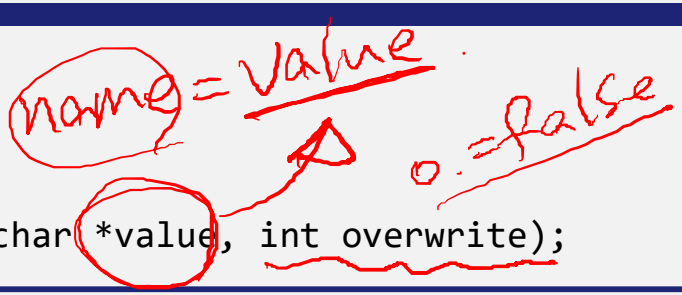
Example – Process ID

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

int main() {
  int pid= 0;
  if ((pid = fork()) == 0) {
    sleep(1);
    printf("My pid %d\n", getpid());
    printf("My Parent pid %d\n", getppid());
  } else if(pid > 0) {
    printf("Parent pid %d\n", getpid());
    sleep(5);
  } else {
    perror("fork error :");
    exit(0);
  }
  return 0;
}
```

```
Parent pid 109
My pid 110
My Parent pid 109
```

Environment 관련 System Calls

```
#include <stdlib.h>
char *getenv(const char *name);
int putenv(char *string);
int setenv(const char *name, const char *value, int overwrite);
```



- getenv()
  - name 이름을 갖는 환경 변수에 저장된 값을 읽어 옴
  - 만약 일치하는 name을 갖는 환경 변수가 있다면 "값"을 반환, 없다면 NULL 반환
- putenv()
  - 환경변수를 추가하거나, 기존 환경 변수의 값 변경
  - 성공할 경우 0, 실패할 경우 -1 반환
- setenv()
  - name이 존재하지 않을 경우, value 값을 가지는 name 환경변수 추가
  - name이 존재하고 overwrite가 0이 아니라면 값을 value로 변경
  - name이 존재하고 overwrite가 0이면 값은 바뀌지 않음
  - 성공할 경우 0, 환경변수 공간이 충분치 않다면 -1 반환

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

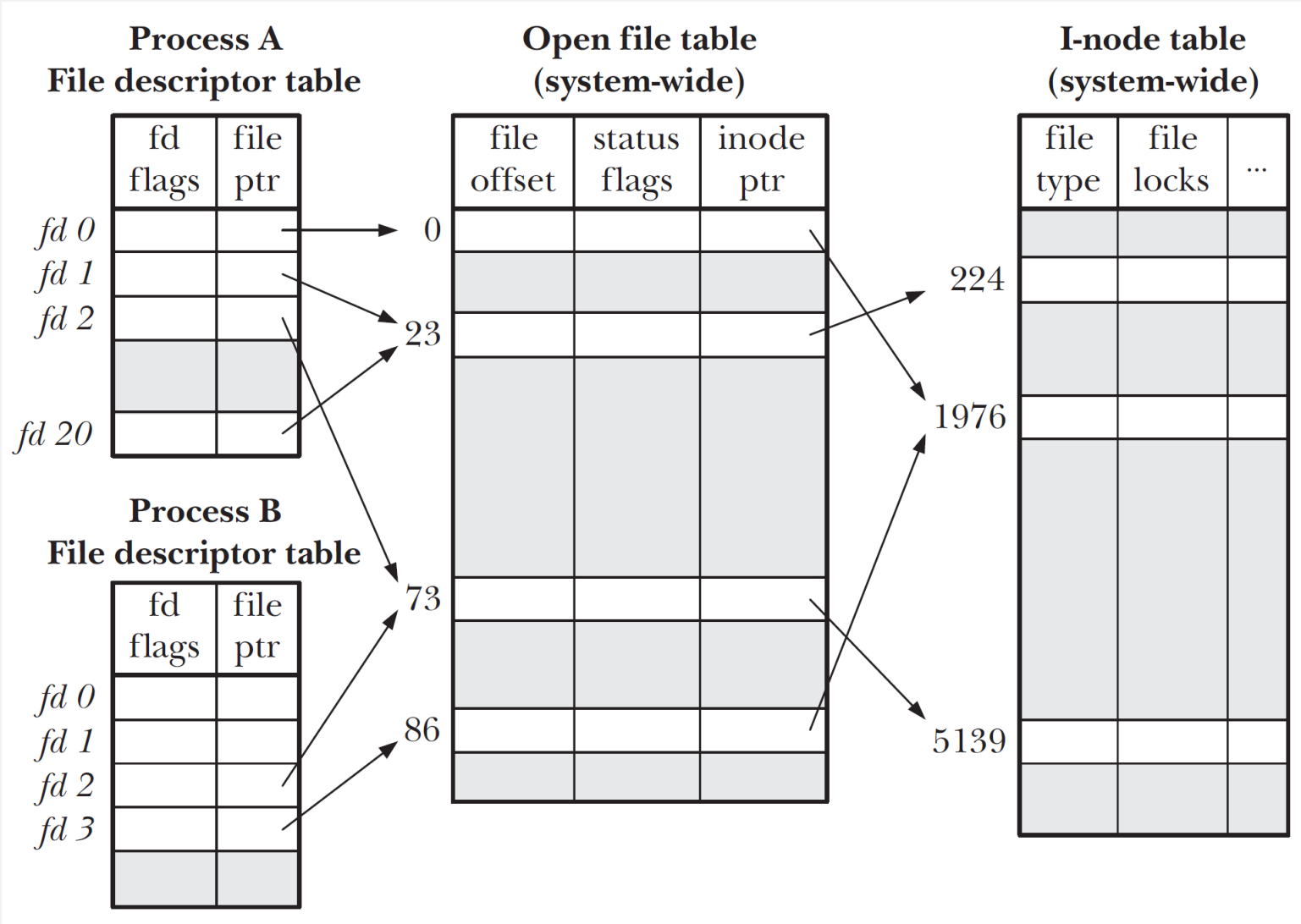
Example – Environment

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

extern char **environ;
int main(int argc, char *argv[], char *envp[]) {
  int i;
  for (i = 0; argv[i] != NULL; i++)
    printf("argv[%d] = %s\n", i, argv[i]);
  printf("\n");
  for (i = 0; envp[i] != NULL; i++)
    printf("envp[%d] = %s\n", i, envp[i]);
  printf("\n");
  for (i = 0; environ[i] != NULL; i++)
    printf("environ[%d] = %s\n", i, environ[i]);
  printf("\nHOME= %s", getenv("HOME"));
  putenv("HOME=/home/user01/test");
  printf("\nHOME= %s\n", getenv("HOME"));
  return 0;
}
```

```
environ[12] = LD_PRELOAD=/usr/local/lib/repl.so
environ[13] = DISPLAY=MAGIC
environ[14] = HOME=/home/runner
environ[15] = TERM=xterm-256color

HOME= /home/runner
HOME= /home/user01/test
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec-and-file (1/5)



- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec-and-file (2/5)

```
// openfexec.c 파일을 만들고 아래 내용 작성
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv) {
  int fd= atoi(argv[1]);  /* I know it is 3 */
  long pos;
  pos = lseek(fd, 0l, SEEK_CUR);
  printf("\tPosin openfexec(): is %ld\n", pos);
  /* an arbitrary number */
  pos = lseek(fd, 50l, SEEK_CUR);
  printf("\tNewpos after lseek() in openfexec() is %ld\n", pos);
  /* return non-zero if not OK */
  return (pos < 0 ? !0 : 0);
}
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec-and-file (3/5)

```
// main.c 파일에 아래 내용 작성
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
  int fdes, pid, cflag, excode;
  long pos;
  char sfdes[10];

  /* open its srcfile */
  fdes = open("main.c", O_RDONLY);
  sprintf(sfdes, "%d", fdes);
  /* print file descriptor */
  printf("fdes = %d\n", fdes);
  /* 20 is an arbitrary number */
  pos = lseek(fdes, 20l, SEEK_SET);
  printf("Current position before fork() is %ld\n", pos);
  if(!fork()) {   /* child */
    /* 40 is also arbitrary */
    pos = lseek(fdes, 40l, SEEK_SET);
    printf("Current position in child after fork() is %ld\n", pos);
  }
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec-and-file (4/5)

```
else {     /* parent */
  /* wait for 1st child to terminate */
  wait((int *)0);
  /* to get pos in file */
  pos = lseek(fdes, 0l, SEEK_CUR);
  printf("Current position in parent after fork() is %ld\n", pos);
  /* fork again --child process */
  if (!fork()) {
    /* overlay with another program */
    execl("./openfexec", "openfexec", sfdes, (char *)0);
    printf("Itis an error to print this line out\n");
  }

  /* parent --no need for else */
  wait(&excode);
  /* exit code is needed */
  pos = lseek(fdes, 0l, SEEK_CUR);
  printf("Current pos in parent after exec() is %ld\n", pos);
  printf("Exit code of a child = %d\n", WEXITSTATUS(excode));
  /* 3rd arg can be any in there */
  cflag = fcntl(fdes, F_GETFD, 0);
  printf("close-on-exec flag = %d\n", cflag);
  /* set close-on-exec flag */
  fcntl(fdes, F_SETFD, 1);  // 1 = set
```

- Process의 개념
- fork()
- exec()
- exit()
- wait()
- waitpid()
- 프로세스ID
- 환경변수
- exec와 파일 상속

Example – exec-and-file (5/5)

```
    /* fork again --parent process */
    if ((pid = fork()) != 0) {
      /* wait for a specific child to end */
      waitpid(pid, &excode, 0);
      printf("Exit code of a specific child = %d\n", WEXITSTATUS(excode));
      /* parent terminates */
      exit(0);
    }

    /* child executes another program */
    execl("./openfexec", "openfexec", sfdes, (char *)0);
    printf("It is an error to print this line out\n");
  }
}
```

```
fdes = 3
Current position before fork() is 20
Current position in child after fork() is 40
Current position in parent after fork() is 40
    Posin openfexec(): is 40
    Newpos after lseek() in openfexec() is 90
Current pos in parent after exec() is 90
Exit code of a child = 0
close-on-exec flag = 0
    Posin openfexec(): is -1
    Newpos after lseek() in openfexec() is -1
Exit code of a specific child = 1
```

실습문제

- **`execve()`를 이용하여 `execlp()`를 구현하시오.**
  - 기존 `execlp()`와 겹치지 않도록 `myexeclp()` 함수로 구현할 것
  - `execlp()`는 인자의 길이가 가변적 임
    - 가변적 인자를 처리해주기 위해서는 `stdarg(3)`이 필요함
    - 인터넷에서 사용 예 검색해볼 것
    - 구현의 편의를 위해 인자의 최대 길이는 MAXARGS를 사용할 것
  - 환경변수 `PATH`로부터 경로명을 가져와 테스트해 볼 것
    - 파일이 있는지, 그 파일이 실행 가능한지 테스트해야 함
    - 제일 뒤의 경로명부터 테스트해야 함
    - 처음 실행할 수 있는 경로명에 있는 실행파일을 사용할 것
  - 실패 시 -1 반환

THANK YOU

Presented by Hasoo Eun