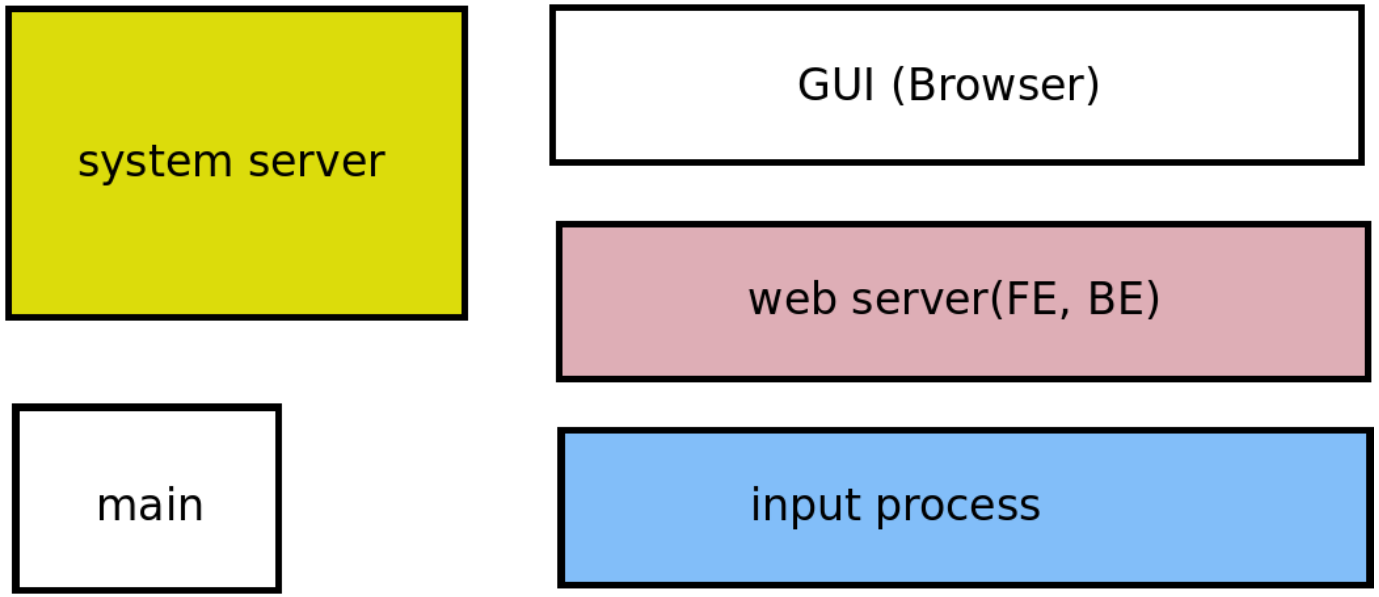
프로세스 관련 시스템 콜

목적

- 프로세스의 개념

- 프로세스 관련 시스템 콜 및 사용 방법 학습
  - 생성 / 종료/ 대기 / 실행 / 응용

토이 프로젝트 실습 구현


system server
GUI (Browser)
web server(FE, BE)
main
input process

프로세스와 프로그램

- 실행 중인 프로그램

- 하나의 프로그램으로 여러 프로세스 생성 가능

프로세스

- 격리(isolation)
- 다중처리(Multiplexing)
- 공유(Interaction or share)

프로세스 ID

- getpid() 시스템 콜

프로세스의 메모리 레이아웃

Listing 6-1: Locations of program variables in process memory segments

proc/mem_segments.c

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

char globBuf[65536];            /* Uninitialized data segment */
int primes[] = { 2, 3, 5, 7 };  /* Initialized data segment */

static int
square(int x)                   /* Allocated in frame for square() */
{
    int result;                 /* Allocated in frame for square() */

    result = x * x;
    return result;              /* Return value passed via register */
}

static void
doCalc(int val)                 /* Allocated in frame for doCalc() */
{
    printf("The square of %d is %d\n", val, square(val));

    if (val < 1000) {
        int t;                  /* Allocated in frame for doCalc() */

        t = val * val * val;
        printf("The cube of %d is %d\n", val, t);
    }
}

int
main(int argc, char *argv[])    /* Allocated in frame for main() */
{
    static int key = 9973;      /* Initialized data segment */
    static char mbuf[10240000]; /* Uninitialized data segment */
    char *p;                    /* Allocated in frame for main() */

    p = malloc(1024);           /* Points to memory in heap segment */

    doCalc(key);

    exit(EXIT_SUCCESS);
}
```

proc/mem_segments.c

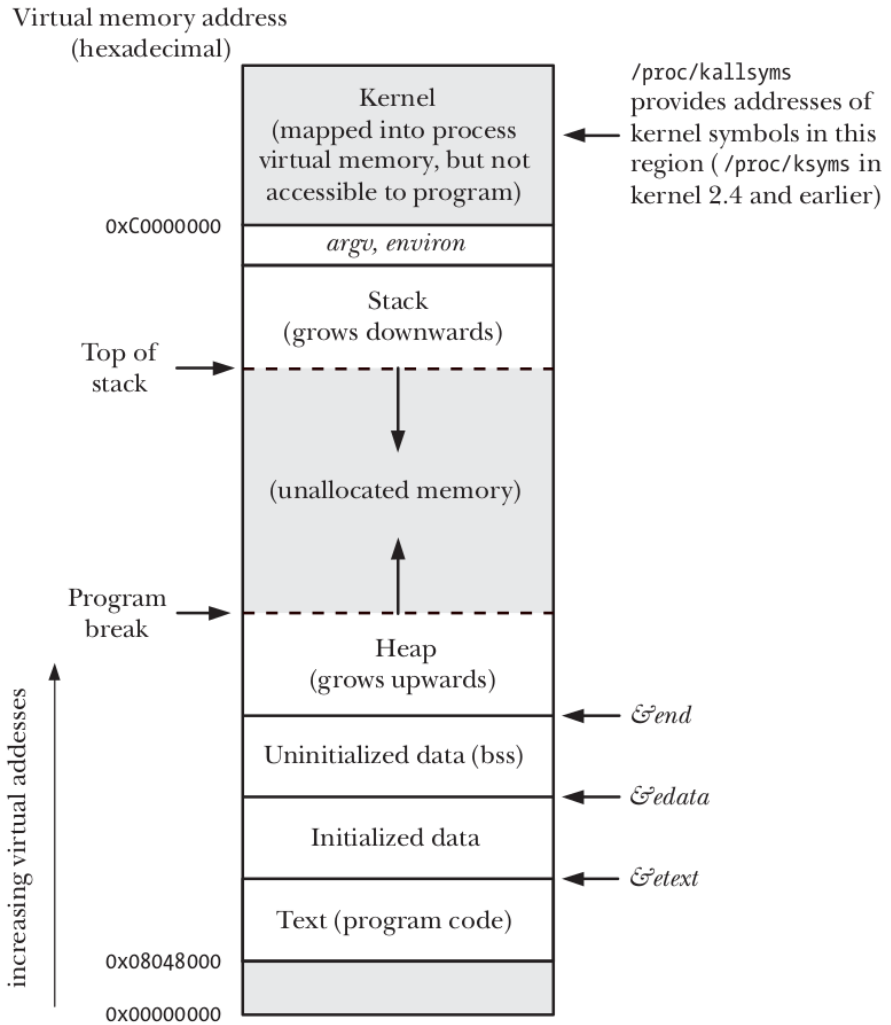


Figure 6-1: Typical memory layout of a process on Linux/x86-32

스택과 스택 프레임

- 스택에는 다음과 정보가 담겨 있음.
- 함수 인자와 지역 변수
- 호출 연결 정보

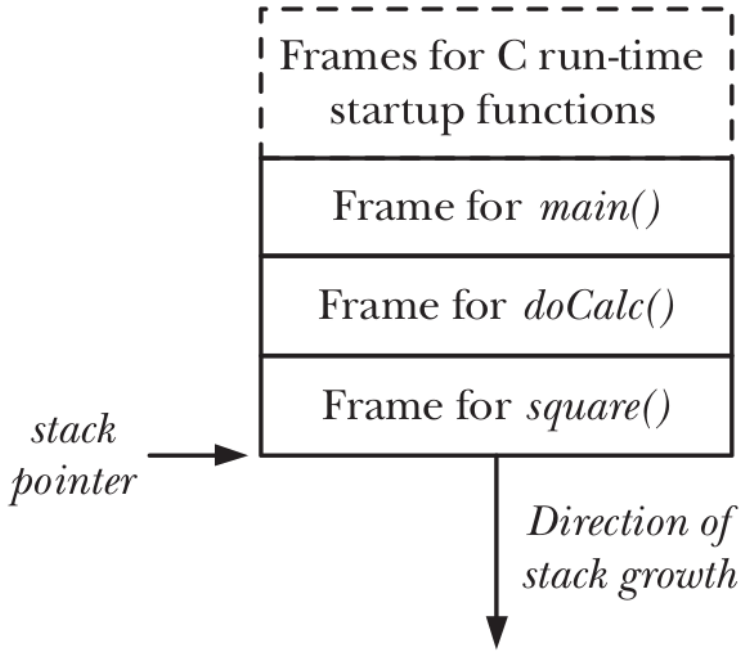


Figure 6-3: Example of a process stack

가상 메모리 개요.

- 우리가 보는 메모리는? 가상 메모리
- 페이징 시스템
  - 직접 접근 X
  - 페이지 테이블을 통해 접근
- 자세한 내용은 운영체제 시간에...

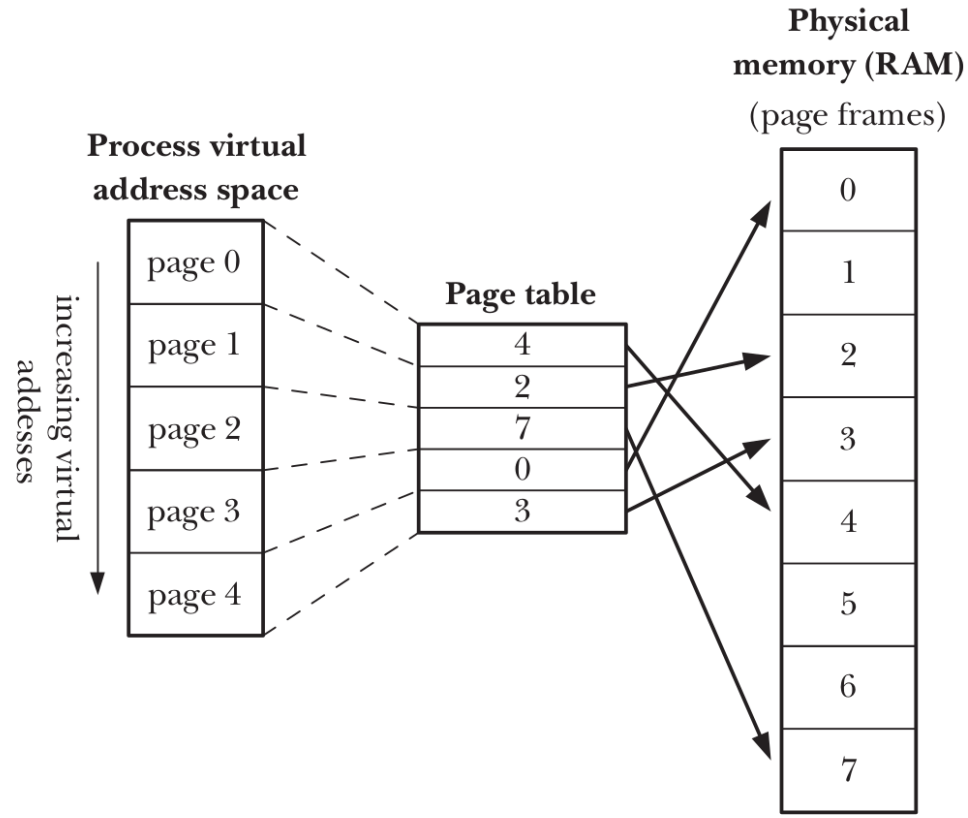


Figure 6-2: Overview of virtual memory

명령행 인자(argc, argv)

Listing 6-2: Echoing command-line arguments

proc/necho.c

```
#include "tlpi_hdr.h"

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

    for (j = 0; j < argc; j++)
        printf("argv[%d] = %s\n", j, argv[j]);

    exit(EXIT_SUCCESS);
}
```

proc/necho.c

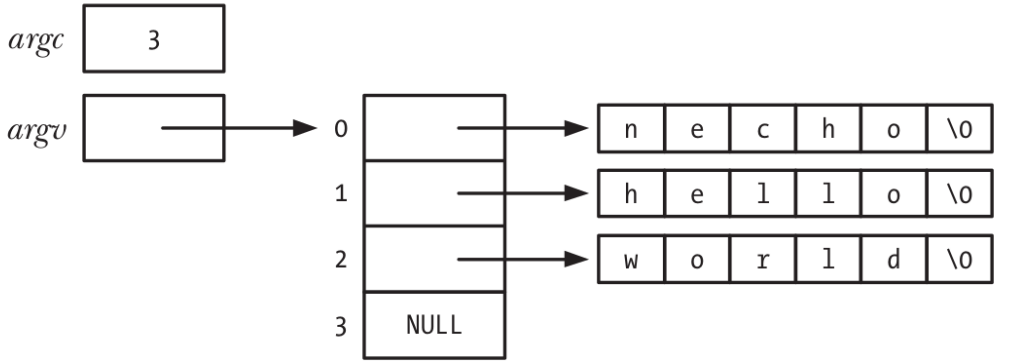


Figure 6-4: Values of *argc* and *argv* for the command *necho hello world*

환경 변수 목록

- 유닉스 전통적인 Key/Value
- export SHELL=/bin/bash
- printenv
- setenv
- getenv
- 주로 부팅 시 설정 값 활용하기 위해 사용

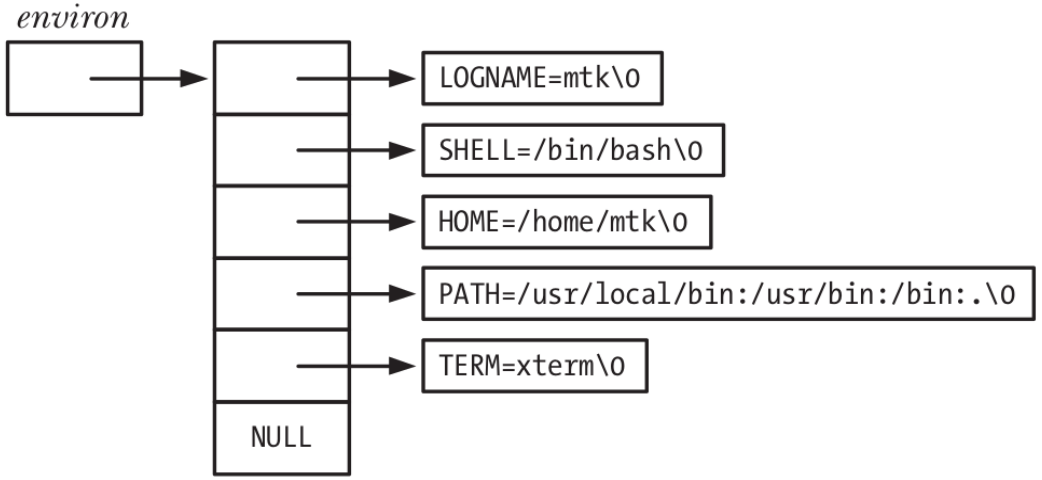


Figure 6-5: Example of process environment list data structures

환경 변수 목록

Listing 6-3: Displaying the process environment

proc/display_env.c

```
#include "tlpi_hdr.h"

extern char **environ;

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

    for (ep = environ; *ep != NULL; ep++)
        puts(*ep);

    exit(EXIT_SUCCESS);
}
```

proc/display_env.c

The *getenv()* function retrieves individual values from the process environment.

```
#include <stdlib.h>

char *getenv(const char *name);
```

Returns pointer to (value) string, or NULL if no such variable

프로세스 관련 시스템 콜

- fork()
  - 프로세스 생성
- exit()
  - 프로세스 종료
- wait()
  - 자식 프로세스 대기
  - 대기하는 이유는?
- execve()
  - 프로그램 실행(실행 파일 메모리 로드)

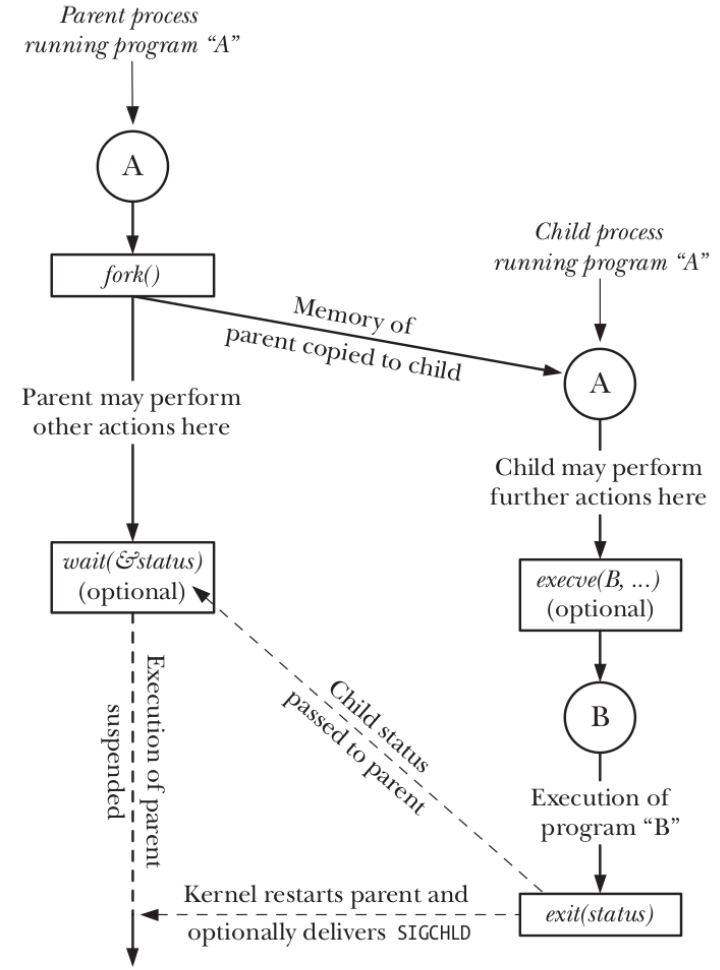


Figure 24-1: Overview of the use of *fork()*, *exit()*, *wait()*, and *execve()*

프로세스 생성

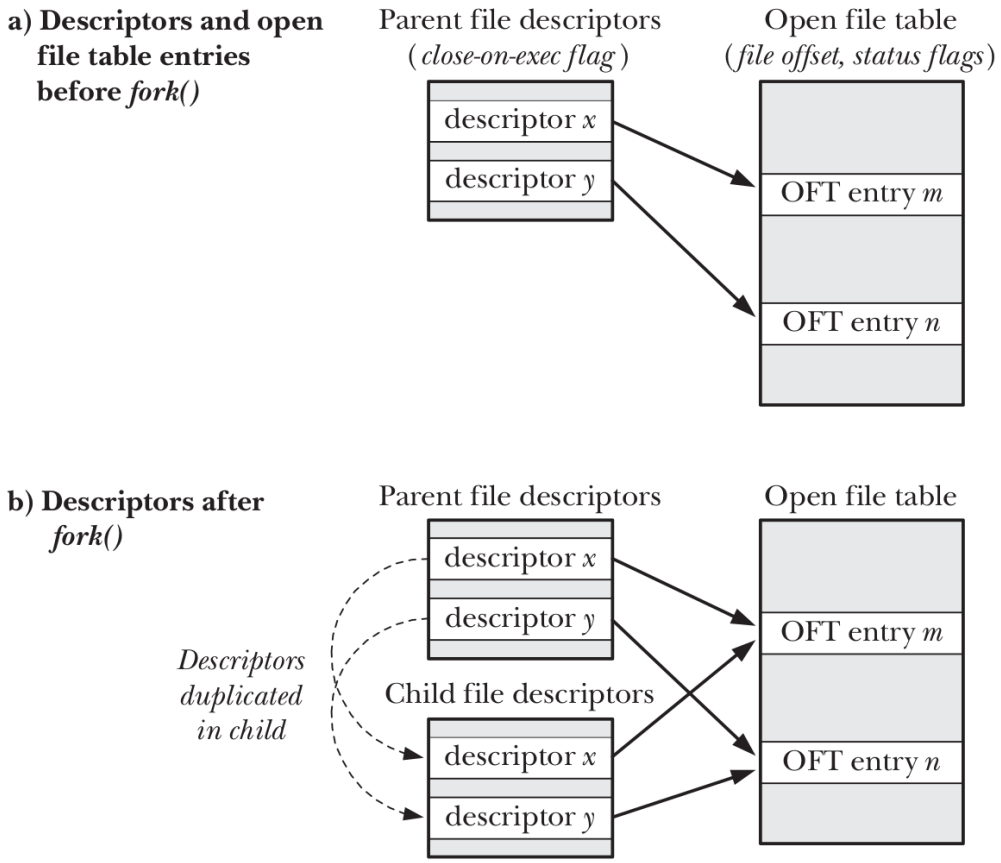
```
#include <unistd.h>

pid_t fork(void);
```

In parent: returns process ID of child on success, or −1 on error;
in successfully created child: always returns 0

procexec/t_fork.c

```
————————————————————————————————————————————————— procexec/t_fork.c
#include "tlpi_hdr.h"

static int idata = 111;             /* Allocated in data segment */

int
main(int argc, char *argv[])
{
    int istack = 222;               /* Allocated in stack segment */
    pid_t childPid;

    switch (childPid = fork()) {
    case -1:
        errExit("fork");

    case 0:
        idata *= 3;
        istack *= 3;
        break;

    default:
        sleep(3);                   /* Give child a chance to execute */
        break;
    }

    /* Both parent and child come here */

    printf("PID=%ld %s idata=%d istack=%d\n", (long) getpid(),
            (childPid == 0) ? "(child) " : "(parent)", idata, istack);

    exit(EXIT_SUCCESS);
}
————————————————————————————————————————————————— procexec/t_fork.c
```

fork 와 파일 디스크립터



Copy On Write

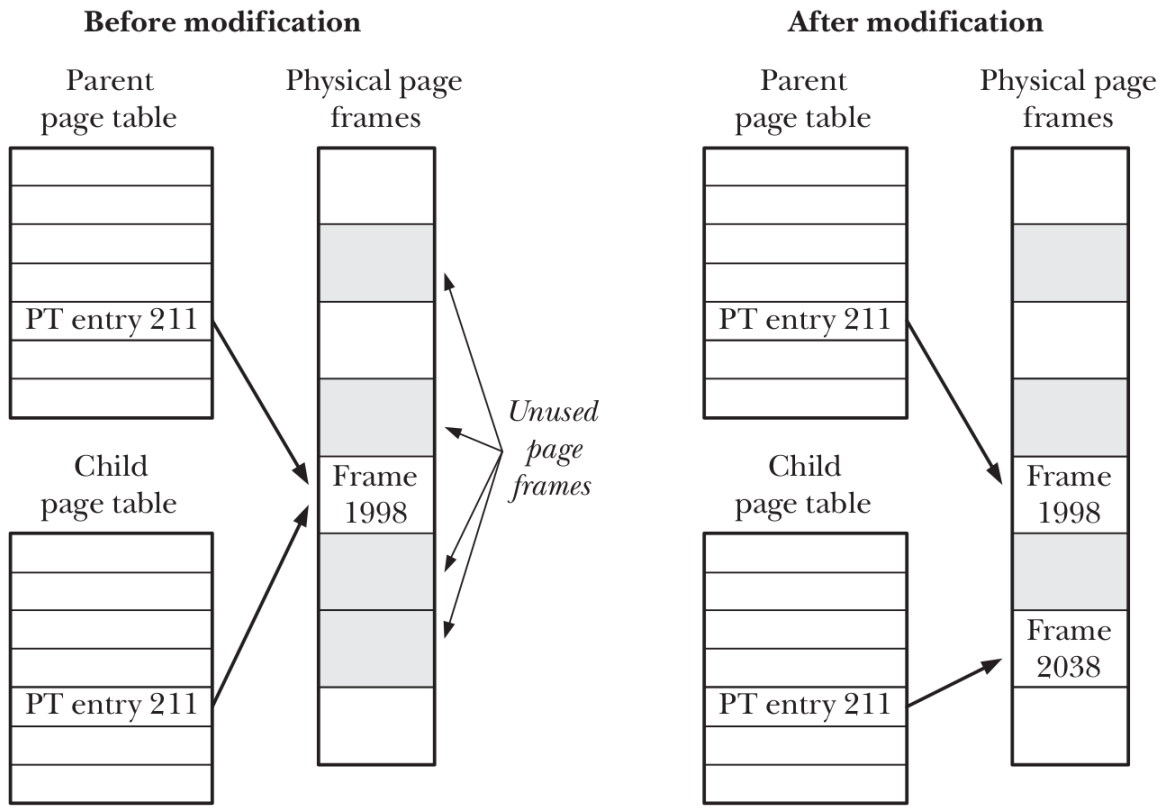


Figure 24-3: Page tables before and after modification of a shared copy-on-write page

프로세스 종료

```
#include <stdlib.h>

void exit(int status);
```

자식 프로세스 감시

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

pid_t wait(int *status);
```

Returns process ID of terminated child, or −1 on error

자식 프로세스 감시

```
int
main(int argc, char *argv[])
{
    int numDead;       /* Number of children so far waited for */
    pid_t childPid;    /* PID of waited for child */
    int j;

    if (argc < 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s sleep-time...\n", argv[0]);

    setbuf(stdout, NULL);           /* Disable buffering of stdout */

    for (j = 1; j < argc; j++) {    /* Create one child for each argument */
        switch (fork()) {
        case -1:
            errExit("fork");

        case 0:                     /* Child sleeps for a while then exits */
            printf("[%s] child %d started with PID %ld, sleeping %s "
                    "seconds\n", currTime("%T"), j, (long) getpid(), argv[j]);
            sleep(getInt(argv[j], GN_NONNEG, "sleep-time"));
            _exit(EXIT_SUCCESS);

        default:                    /* Parent just continues around loop */
            break;
        }
    }

    numDead = 0;
    for (;;) {                      /* Parent waits for each child to exit */
        childPid = wait(NULL);
        if (childPid == -1) {
            if (errno == ECHILD) {
                printf("No more children - bye!\n");
                exit(EXIT_SUCCESS);
            } else {                /* Some other (unexpected) error */
                errExit("wait");
            }
        }

        numDead++;
        printf("[%s] wait() returned child PID %ld (numDead=%d)\n",
                currTime("%T"), (long) childPid, numDead);
    }
}
```

---------- procexec/multi_wait.c

waitpid()

- 특정 프로세스 기다리기

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

pid_t waitpid(pid_t pid, int *status, int options);
                                    Returns process ID of child, 0 (see text), or –1 on error
```

- 에러 처리
  - WIFEXITED(status)
  - WIFSIGNALED(status)
  - WIFSTOPPED(staus)

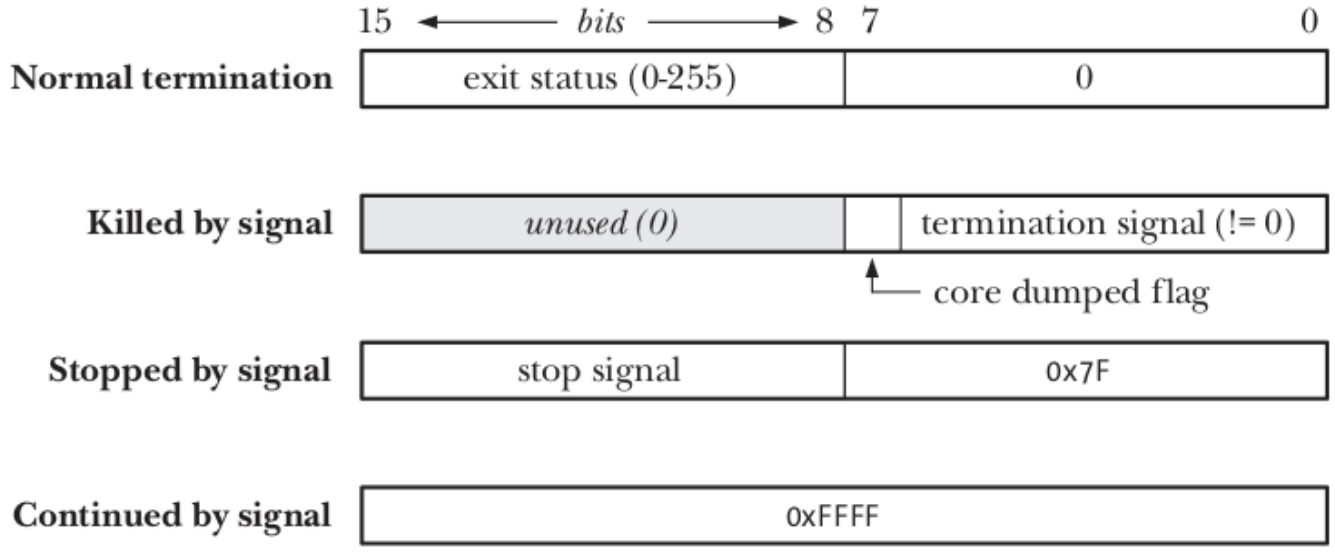


Figure 26-1: Value returned in the *status* argument of *wait()* and *waitpid()*

wait 상태 값 출력

```
void                    /* Examine a wait() status using the W* macros */
printWaitStatus(const char *msg, int status)
{
```



```
    if (msg != NULL)
        printf("%s", msg);

    if (WIFEXITED(status)) {
        printf("child exited, status=%d\n", WEXITSTATUS(status));

    } else if (WIFSIGNALED(status)) {
        printf("child killed by signal %d (%s)",
                WTERMSIG(status), strsignal(WTERMSIG(status)));
#ifdef WCOREDUMP        /* Not in SUSv3, may be absent on some systems */
        if (WCOREDUMP(status))
            printf(" (core dumped)");
#endif
        printf("\n");

    } else if (WIFSTOPPED(status)) {
        printf("child stopped by signal %d (%s)\n",
                WSTOPSIG(status), strsignal(WSTOPSIG(status)));

#ifdef WIFCONTINUED     /* SUSv3 has this, but older Linux versions and
                           some other UNIX implementations don't */
    } else if (WIFCONTINUED(status)) {
        printf("child continued\n");
#endif

    } else {            /* Should never happen */
        printf("what happened to this child? (status=%x)\n",
                (unsigned int) status);
    }
}
```

procexec/print_wait_status.c

프로그램 실행

```
#include <unistd.h>

int execve(const char *pathname, char *const argv[], char *const envp[]);
                                        Never returns on success; returns –1 on error
```

Listing 27-1: Using *execve()* to execute a new program

```
———————————————————————————————————— procexec/t_execve.c
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    char *argVec[10];           /* Larger than required */
    char *envVec[] = { "GREET=salut", "BYE=adieu", NULL };

    if (argc != 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s pathname\n", argv[0]);

    argVec[0] = strrchr(argv[1], '/');      /* Get basename from argv[1] */
    if (argVec[0] != NULL)
        argVec[0]++;
    else
        argVec[0] = argv[1];
    argVec[1] = "hello world";
    argVec[2] = "goodbye";
    argVec[3] = NULL;           /* List must be NULL-terminated */

    execve(argv[1], argVec, envVec);
    errExit("execve");          /* If we get here, something went wrong */
}
———————————————————————————————————— procexec/t_execve.c
```

```
$ ./t_execve ./envargs
argv[0] = envargs
argv[1] = hello world
argv[2] = goodbye
environ: GREET=salut
environ: BYE=adieu
```

All of the output is printed by envargs

exec() 라이브러리 함수

```
#include <unistd.h>

int execle(const char *pathname, const char *arg, ...
                /* , (char *) NULL, char *const envp[] */ );
int execlp(const char *filename, const char *arg, ...
                /* , (char *) NULL */);
int execvp(const char *filename, char *const argv[]);
int execv(const char *pathname, char *const argv[]);
int execl(const char *pathname, const char *arg, ...
                /* , (char *) NULL */);
```

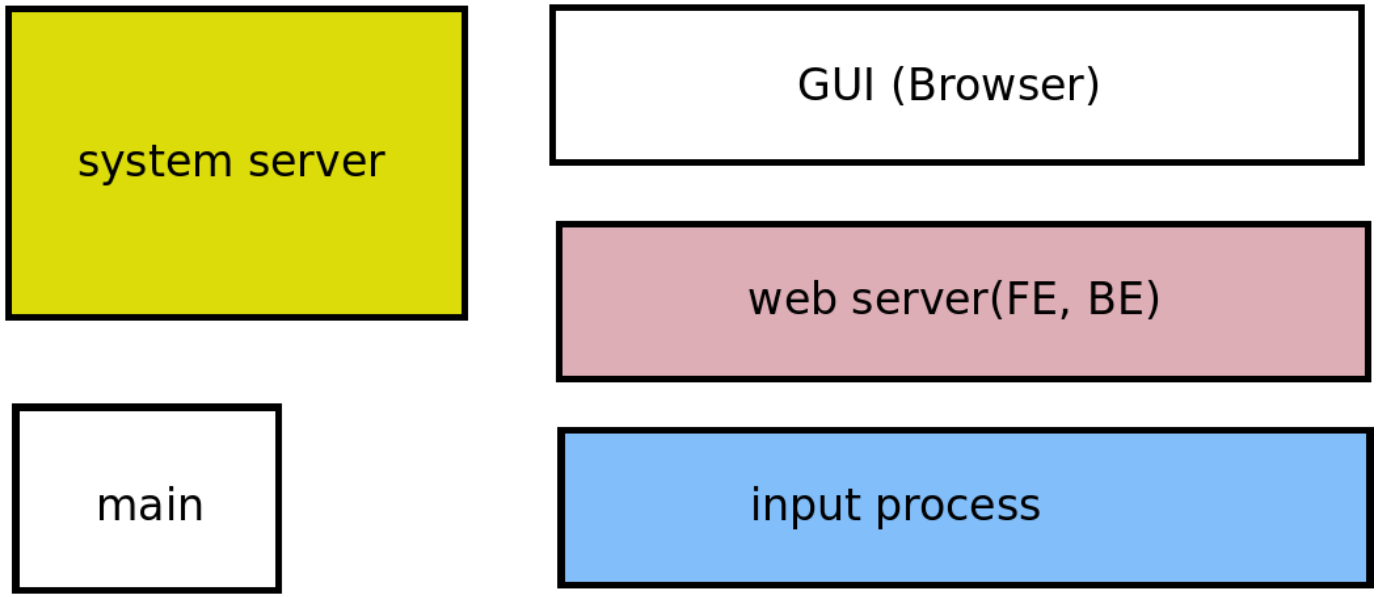
None of the above returns on success; all return −1 on error

실습 코드 분석

- vscode debugger로 디버깅
- tlpi-dist/proc/mem_segments.c
  - 코드 분석 및 실행
- tlpi-dist/procexec/multi_wait.c
  - 코드 분석 및 실행
- tlpi-dist/procexec/t_fork.c
  - 코드 분석 및 실행
- tlpi-dist/procexec/t_execve.c
  - 코드 분석 및 실행

토이 프로젝트 - fork /exec

- fork/exec 구현 실습



토이 프로젝트

- 사전 설치 작업
  - filebrowser 설치
  - google chrome 설치

- filebrowser
  - curl -fsSL https://raw.githubusercontent.com/filebrowser/get/master/get.sh | bash filebrowser -r /path/to/your/files