OS HW 2 - Paging

블랙보드 제출일 : 2019. 12.06

2016320272 권영진

목차

* 개요

앞서 스케쥴러와 같이 자식 프로세스를 message(IPC) 와 signal 을 이용하여 페이징을 구현한다. 부모 프로세스(커널)는 10개의 자식 프로세스를 생성하는데, 생성된 프로세서들끼리 메세지와 신호시스템을 통하여 메모리 접근을 소프트웨어적으로 시도해 볼 것이다.

* 개발환경

가상머신 : Oracle Virtual Box
클라이언트 : Ubuntu 16.04.6 LTS (64bit) with linux 4.4
하드웨어 스펙 : Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz (CPU 4 core), 8GB(RAM)
가상머신 할당스펙 : CPU 4 core, 4GB RAM
사용언어 : C++/C

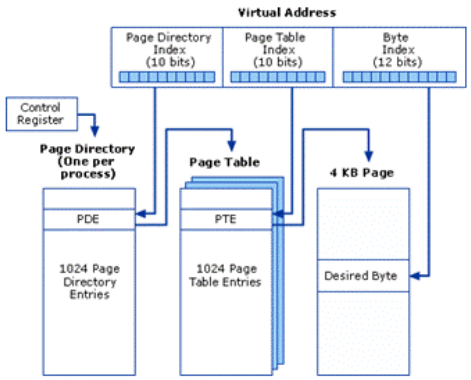
* 코드 이해를 위한 배경 지식

Demand Paging

필요한 페이지만 메모리로 가져오는 것을 'Demand Paging'이라고 한다. 실제 프로그램이 실행 중일 때 코드가 수십 수만줄에 넘어가면 모든 데이터들이 메모리에 로드되기는 어렵다. 즉 프로세스가 전부 올라가는 것이 아닌 page 단위로 필요할 만한 것이 (on demand) 메모리에 올라가는 것이다. 이러한 방식으로 메모리에 데이터가 적재되다가 이후 메모리가 가득찼을 때 디스크에 데이터를 보내고 (swap out) 다른 데이터를 들여 오는 것 (swap in) 을 'swapping' 이라고한다. 이 때 적절한 알고리즘에 따라(Least Recently Used) 교체된다.

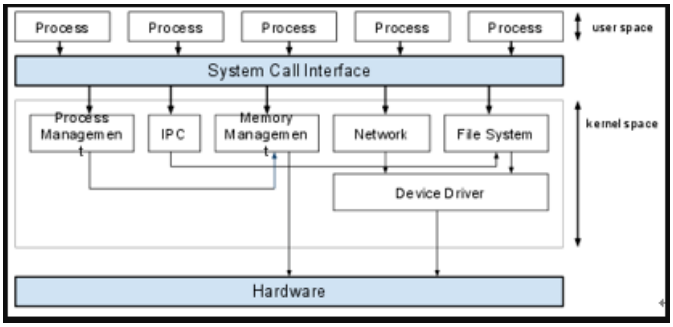
Multi Level Paging

페이징이란 선형 논리 주소를 물리 주소로 변환하는 과정이다. 가상 주소의 경우 기본적으로 page number 와 page offset을 이용하여 실제 물리 주소에 접근할 수 있는데 single level 에서는 4GB 주소 공간을 표현하기 위해 20 비트의 PTE가 필요하기 때문에 너무 커지는 페이지 테이블의 크기를 줄이기 위하여 x86 에서는 Page directory와 Page table을 나누어서 처리한다. MMU는 가상주소의 상위 10bit으로 PDE를 구하여 페이지 테이블에 접근하고 이후 페이지에 접근하면 실제 페이지내에서 offset을 이용하여 최종적인 물리 주소를 접근할 수 있다.



IPC(Inter Process Comunication)

수많은 객체로 이루어진 Linux 시스템이 원할히 작동 하기 위해서는 각 객체간의 원할한 "통신"이 필수적이다. 프로세스가 파일을 열기 위해서는 일련의 작업을 위해서 커널과 통신을 해야할것이다. 특정 프로세스가 가지는 데이타를 다른 프로세스에게 넘겨서 이를 처리하도록 하는 작업도 필요하다. 이러한 데이타 통신을 위해서 Linux에서는 IPC 라는 내부 프로세스간 통신을 위한 도구를 사용한다.



[리눅스 커널 구조] : IPC example

프로세스간 데이터를 통신할때는 pipe, semaphore 등 다양한 방법이 존재한다. 이번 보고서에서는 System V message queue 방식을 사용할 것인데, 이는 다른 방식에 비하여 구현이 직관적이며 메시지 큐에 사용하고 싶은 데이터에 번호를 붙임으로써 여러 개의 프로세스가 동시에 데이터를 쉽게 다룰 수 있기 때문이다. 실제 코드는 다음과 같다.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key_t key, int msgflg);
// key_t key : 다른 큐와 구별하기 위한 번호.
// int msgflg : 큐생성 옵션, 이번 보고서에서는 IPC_CREAT 사용.
// IPC_CREAT :  key에 해당하는 큐가 있다면 큐의 식별자를 반환하며, 없으면 생성.
int msgsnd (int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);
// int msqid : 메시지 큐 식별자로서 msgget 함수의 리턴값이다.
// void *msgp : 전송할 자료, 원하는 자료형을 사용할 수 있다.
// size_t msgsz : 전송할 자료의 크기, sizeof(자료형) 사용한다.
// int msgflg : 동작 옵션, 큐에 공간이 있을때까지 기다리면 0,
// 여유공간 없으면 복귀 시킬경우, IPC_NOWAIT.
ssize_t msgrcv (int msqid, struct  msgbuf  *msgp,  size_t msgsz,
                long msgtyp, int msgflg);
// logn msgtyp : 수신할 메시지 타입, 0 보다 크면 원하는 타입 지정가능
// int msgflg : 동작 옵션
```

* 실제 코드 분석(paging.c)

- Headerfile & Define Constant

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>

#define CHILDNUM 10
#define QUANTUM 3
#define PDIRNUM 1024
#define PTLBNUM 1024
#define FRAMENUM 1024
#define QNUM 128
```

IPC(message) 및 signal 등을 사용하기 위한 헤더파일을 선언한다. 편의를 위하여 자식 프로세스의 수 및 퀀텀을 미리 지정한다. 페이지 디렉토리와 페이지 테이블의 경우 2^10의 개수를 가지고 있다.

- Data Structure & Functions

```
// Data Structure & Functions
typedef struct{
  int valid;
  int pfn;
}TABLE;

typedef struct{
  int valid;
  TABLE*  pt;
}PDIR;

struct msg_buffer{
  long int  mtype;
  int pid_index;
  unsigned int  vm[10];
};

FILE* fp;
pid_t pid[CHILDNUM];
PDIR pdir[CHILDNUM][PDIRNUM];
struct msg_buffer msg;
int cpu_burst[CHILDNUM] ={2,2,4,4,6,6,8,8,5,5};
int cpu_burst_ref[CHILDNUM];
int rq[QNUM]; //RUNQUEUE
int hd,tl = 0;
int free_page_list[1024] ; //FREE PAGE LIST
int fpl_tl,fpl_hd = 0;
int idx, total_tik, count, proc_done = 0;
int msgq, msgret; // for IPC msg
int key = 0x12345;

void kernelHandler(int sig);
void processHandler(int sig);
void toFreePageList(PDIR* p_dir);
```

사용할 테이블과 메세지에 사용될 자료구조를 구조체의 형태로 정의하였다.

- Main & Functions

전체적인 흐름

1. 타이머를 설정하여(setitimer) 매 주기 마다 부모프로세스(커널)은 신호를 받는다.
2. 커널이 신호를 받으면 핸들러(kernelHandler) 에 따라서 실행중인 큐에 따라서 해당 프로세스에 시그널을 준다.(kill)
3. 자식 프로세스는 신호를 받으면 해당 핸들러가 실행되어(processHandler) 현재cpu_burst를 감소시키고 가상 주소를 만들고 커널에 메세지를 보낸다(msgsnd). 이 때의 가상주소는 비트연산을 이용하여 페이지디렉토리 10bit, 페이지테이블 10bit, 오프셋 12bit 으로 구성된다.
4. 이후 커널은 메세지를 받을때마다 (msgrcv) 가상주소를 물리주소로 맵핑해주는 역할을 하며 페이지의 참조 여부에 따라 page fault를 발생시키고 valid bit을 변환시켜준다.

이해를 위하여 다음과같은 순서로 코드를 구분한다.

```
1. At first, physical memory has to be fragmented in page size, and OS must maintain the free page frame list
2. One process (parent process) acts as kernel, the other 10 processes act as user process.
3. OS maintains table for each process. OS allocates an empty-initialized page table when it creates a
new process.
4. When a user process gets a time slice (tick), it accesses addresses (pages).
5. A user(child) process sends IPC message that contains memory access request for 10 pages.
6. VA consists of three parts: page directory number, page number and page offset.
7. Then, OS checks the page table. If the page table entry is valid, the physical address is accessed.
8. If the page table entry is invalid, the OS takes a new free page frame from the free page list. Then, the page table is
updated with page frame number.
```

```
int main(int argc, char* argv[])
{
  fp = fopen("paging.txt", "w");
  unsigned int vm[10]; // virtual memory
  unsigned int offset[10]; // vm offset
  unsigned int pdir_idx[10]; // vm page directory
  unsigned int tbl_idx[10]; // vm page table
  int pid_index;
  for (int l = 0; l < CHILDNUM; l++) cpu_burst_ref[l] = cpu_burst[l];
  // 1. At first, physical memory has to be fragmented in page size,
  // and OS must maintain the free page frame list
  for (int l = 0; l < FRAMENUM; l++) {
    free_page_list[l] = l;
    fpl_tl++;
  }
  msgq = msgget(key, IPC_CREAT | 0666);
// 2. One process (parent process) acts as kernel, the other 10 processes act as user process.
while (idx < CHILDNUM) {
  // 3. OS maintains table for each process.
  // OS allocates an empty-initialized page table when it creates a new process.
      for (int i = 0; i < CHILDNUM; i++) {
      for (int j = 0; j < PDIRNUM; j++) {
        pdir[i][j].valid = 0;
        pdir[i][j].pt = NULL;
      }
    }
    pid[idx] = fork();
    rq[(tl++) % QNUM] = idx;
    if (pid[idx] == -1) {
      perror("fork error");
      return 0;
    }
    else if (pid[idx] == 0) {
      // 4. A user(child) process sends IPC message that contains memory access request for 10 pages.
      struct sigaction old_sa;
      struct sigaction new_sa;
      memset(&new_sa, 0, sizeof(new_sa));
      new_sa.sa_handler = &processHandler;
      sigaction(SIGINT, &new_sa, &old_sa);
      while (1);
      return 0;
    }
    else {
      // parent
      struct sigaction old_sa;
      struct sigaction new_sa;
      memset(&new_sa, 0, sizeof(new_sa));
      // 5. When a user process gets a time slice (tick), it accesses addresses (pages).
      new_sa.sa_handler = &kernelHandler;  // 매 시간마다 발생되는 신호
      sigaction(SIGALRM, &new_sa, &old_sa);
      struct itimerval new_itimer, old_itimer;
      new_itimer.it_interval.tv_sec = 0;
      new_itimer.it_interval.tv_usec = 50000;  // 처음 신호까지의 시간
      new_itimer.it_value.tv_sec = 0;
      new_itimer.it_value.tv_usec = 50000;  // 다음 부터 주기적인 신호 시간
      setitimer(ITIMER_REAL, &new_itimer, &old_itimer);  // 타이머 시작
    }
    idx++;
  }

  while (1) {
    msgret = msgrcv(msgq, &msg, sizeof(msg), IPC_NOWAIT, IPC_NOWAIT);

if (msgret != -1) {
      pid_index = msg.pid_index;
      // 6. VA consists of three parts: page directory number, page table number and page offset.
      for (int k = 0; k < 10; k++) {
        vm[k] = msg.vm[k];
        offset[k] = vm[k] & 0xfff;
        tbl_idx[k] = (vm[k] & 0x3ff000) >> 12;
        pdir_idx[k] = (vm[k] & 0xffc00000) >> 20;

      // 7. Then, OS checks the page table. If the page table entry is valid, the physical address is accessed.
      // 8. If the page table entry is invalid, the OS takes a new free page frame from the free page list.
      // Then, the page table is updated with page frame number.
        if (pdir[pid_index][pdir_idx[k]].valid == 0)
        {
          fprintf(fp, "FIRST PAGE FAULT!\n");
          printf("FIRST PAGE FAULT!\n");
          TABLE* table = (TABLE*)calloc(PTLBNUM, sizeof(TABLE));
          pdir[pid_index][pdir_idx[k]].pt = table;
          pdir[pid_index][pdir_idx[k]].valid = 1;
```

```
        }

        TABLE* ptbl = pdir[pid_index][pdir_idx[k]].pt;

        if (ptbl[tbl_idx[k]].valid == 0)
        {
          fprintf(fp, "SECOND PAGE FAULT!\n");
          printf("SECOND PAGE FAULT!\n");

          if (fpl_hd != fpl_tl) {
            ptbl[tbl_idx[k]].pfn = free_page_list[(fpl_hd % FRAMENUM)];
            ptbl[tbl_idx[k]].valid = 1;
            fpl_hd++;
          }
          else {
            for (int k = 0; k < CHILDNUM; k++)
            {
              kill(pid[k], SIGKILL);
            }
            msgctl(msgq, IPC_RMID, NULL);
            exit(0);
            return 0;
          }
        }
        fprintf(
            fp,
            "VA : 0x%08x[PDIR : %d, PTBL : %d, OFFEST : 0x%04x] -> PA:0x%08x\n",
            vm[k], pdir_idx[k], tbl_idx[k], offset[k],
            (ptbl[tbl_idx[k]].pfn << 12) + offset[k]);
        printf(
            "VA : 0x%08x[PDIR : %d, PTBL : %d,OFFSET : 0x%04x] -> PA:0x%08x\n",
            vm[k], pdir_idx[k], tbl_idx[k], offset[k],
            (ptbl[tbl_idx[k]].pfn << 12) + offset[k]);
      }
      memset(&msg, 0, sizeof(msg));
    }
  }

  return 0;
}
```

```
void kernelHandler(int sig)
{
  if (proc_done == 1) {
    toFreePageList(pdir[rq[(hd - 1) % QNUM]]);
    proc_done = 0;
  }
  total_tik++;
  count++;
  if (total_tik >= 10000) {
    for (int k = 0; k < CHILDNUM; k++)
    {
      kill(pid[k], SIGKILL);
    }
    msgctl(msgq, IPC_RMID, NULL);
    exit(0);
  }
  fprintf(fp, "-------------------------------------------------\n");
  fprintf(fp, "Tik Times : %d ", total_tik);
  fprintf(fp, "PID :%d ,REMAINING CPU BURST : %d\n", pid[rq[hd % QNUM]],
          cpu_burst[rq[hd % QNUM]]);
  printf("-------------------------------------------------\n");
  printf("Tik Times %d ", total_tik);
  printf("PID : %d ,REMAINING CPU BURST : %d\n", pid[rq[hd % QNUM]],
         cpu_burst[rq[hd % QNUM]]);
  if ((hd % QNUM) != (tl % QNUM)) {
    cpu_burst[rq[hd % QNUM]]--;
    kill(pid[rq[hd % QNUM]], SIGINT); // user process 에 siganl 보낸다.
    if ((count == QUANTUM) | (cpu_burst[rq[hd % QNUM]] == 0)) {
      count = 0;
      if (cpu_burst[rq[hd % QNUM]] != 0) rq[(tl++) % QNUM] = rq[hd % QNUM];
      if (cpu_burst[rq[hd % QNUM]] == 0) {
        cpu_burst[rq[hd % QNUM]] = cpu_burst_ref[rq[hd % QNUM]];
        rq[(tl++) % QNUM] = rq[hd % QNUM];
        proc_done = 1; // 다음 프로세스로 넘어가기 위한 플래그
      }
      hd++;
    }
```

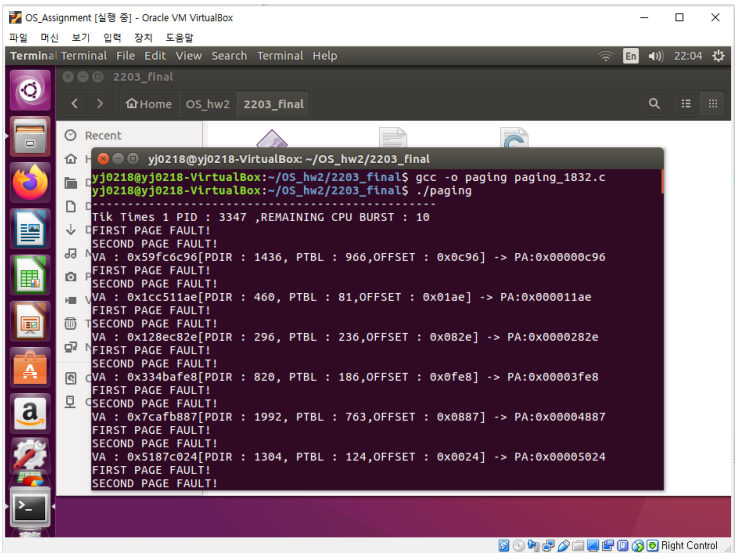
```
  }
}
```

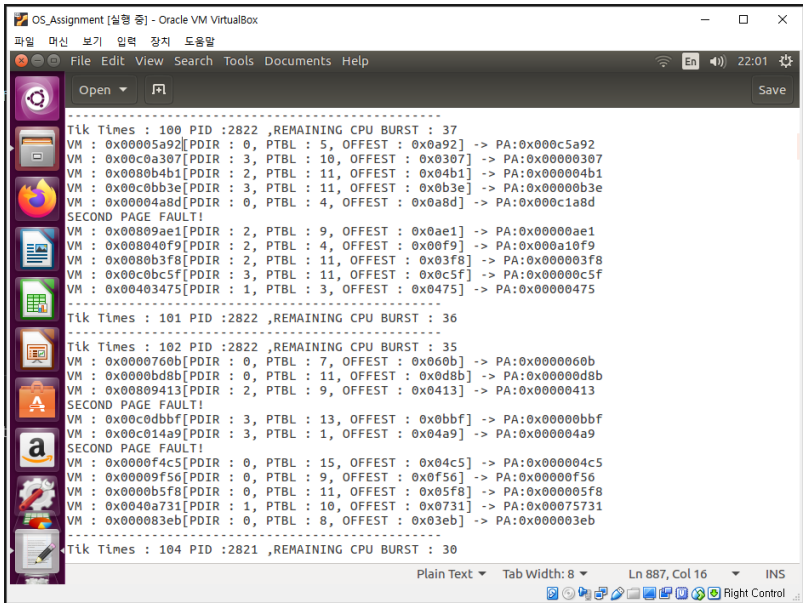
```
void processHandler(int sig)
{
  cpu_burst[idx]--;  // 현재 실행중인 프로세스 cpu burst time 감소
  if (cpu_burst[idx] <= 0) cpu_burst[idx] = cpu_burst_ref[idx];
  memset(&msg, 0, sizeof(msg));
  msg.mtype = IPC_NOWAIT;
  msg.pid_index = idx;
  unsigned int temp_add;
  // 5. A user(child) process sends IPC message that contains memory access request for 10 pages.
  for (int k = 0; k < 10; k++) {
    // 10개의 메모리 접근 시도
    temp_add = (rand() % 1024) << 22;  // 10 bit for directory page table
    temp_add |= (rand() % 1024) << 12;  // 10 bit for page table
    temp_add |= (rand() % 0xfff);  // 12 bit for offset
    msg.vm[k] = temp_add;
  }
  // 4. When a user process gets a time slice (tick), it accesses addresses (pages).
  msgret = msgsnd(msgq, &msg, sizeof(msg), IPC_NOWAIT);
  if (msgret == -1) perror("msgsnd error");
}

void toFreePageList(PDIR* p_dir)
{
  PDIR* temp_dir = p_dir;
  for (int i = 0; i < PDIRNUM; i++)
  {
    if (temp_dir[i].valid == 1) {
      temp_dir[i].valid = 0;
      for (int j = 0; j < PTLBNUM; j++)
        if ((temp_dir[i].pt)[j].valid == 1) {
          free_page_list[(fpl_tl++) % FRAMENUM] = (temp_dir[i].pt)[j].pfn;
        }
      temp_dir[i].pt = NULL;
      free(temp_dir[i].pt);
    }
  }
}
```

* 결과화면

make를 통하여 paging 를 생성하고 실행시켜서 그에따른 쉘 화면이 출력되고 텍스트 파일이 생성되었다.





전체 소스코드

```
#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/time.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <errno.h>

#include <time.h>

#define CHILDNUM 10

#define QUANTUM 3

#define PDIRNUM 1024

#define PTLBNUM 1024

#define FRAMENUM 1024

#define QNUM 128

// Data Structure & Functions

typedef struct {
  int valid;

  int pfn;

} TABLE;

typedef struct {
  int valid;
```

```
  TABLE* pt;

} PDIR;

struct msg_buffer {
  long int mtype;

  int pid_index;

  unsigned int vm[10];
};

FILE* fp;

pid_t pid[CHILDNUM];

PDIR pdir[CHILDNUM][PDIRNUM];

struct msg_buffer msg;

int cpu_burst[CHILDNUM] = {10, 6, 5, 2, 8, 4, 3, 2, 7, 4};

int cpu_burst_ref[CHILDNUM];

int rq[QNUM];  // RUNQUEUE

int hd, tl = 0;

int free_page_list[1024];  // FREE PAGE LIST

int fpl_tl, fpl_hd = 0;

int idx, total_tik, count, proc_done = 0;

int msgq;

int msgret;

int key = 0x12345;

void kernelHandler(int sig);

void processHandler(int sig);

void toFreePageList(PDIR* p_dir);

int main(int argc, char* argv[])

{
  fp = fopen("paging.txt", "w");

  unsigned int vm[10];

  unsigned int offset[10];

  unsigned int pdir_idx[10];

  unsigned int tbl_idx[10];

  int pid_index;

  for (int l = 0; l < CHILDNUM; l++) cpu_burst_ref[l] = cpu_burst[l];

  for (int l = 0; l < FRAMENUM; l++) {
    free_page_list[l] = l;

    fpl_tl++;
  }

  msgq = msgget(key, IPC_CREAT | 0666);

  while (idx < CHILDNUM) {
    for (int i = 0; i < CHILDNUM; i++) {
      for (int j = 0; j < PDIRNUM; j++) {
        pdir[i][j].valid = 0;

        pdir[i][j].pt = NULL;
      }
    }

    pid[idx] = fork();

    rq[(tl++) % QNUM] = idx;
```

```
  if (pid[idx] == -1) {
    perror("fork error");

    return 0;

  }

  else if (pid[idx] == 0) {
    // child

    struct sigaction old_sa;

    struct sigaction new_sa;

    memset(&new_sa, 0, sizeof(new_sa));

    new_sa.sa_handler = &processHandler;

    sigaction(SIGINT, &new_sa, &old_sa);

    while (1)
      ;

    return 0;

  }

  else {
    // parent

    struct sigaction old_sa;

    struct sigaction new_sa;

    memset(&new_sa, 0, sizeof(new_sa));

    new_sa.sa_handler = &kernelHandler;  // 매 시간마다 발생되는 신호

    sigaction(SIGALRM, &new_sa, &old_sa);

    struct itimerval new_itimer, old_itimer;

    new_itimer.it_interval.tv_sec = 0;

    new_itimer.it_interval.tv_usec = 50000;  // 처음 신호까지의 시간

    new_itimer.it_value.tv_sec = 0;

    new_itimer.it_value.tv_usec = 50000;  // 다음 부터 주기적인 신호 시간

    setitimer(ITIMER_REAL, &new_itimer, &old_itimer);  // 타이머 시작
  }

  idx++;
}

while (1) {
  msgret = msgrcv(msgq, &msg, sizeof(msg), IPC_NOWAIT, IPC_NOWAIT);

  if (msgret != -1) {
    pid_index = msg.pid_index;

    for (int k = 0; k < 10; k++) {
      vm[k] = msg.vm[k];

      offset[k] = vm[k] & 0xfff;

      tbl_idx[k] = (vm[k] & 0x3ff000) >> 12;

      pdir_idx[k] = (vm[k] & 0xffc00000) >> 20;

      if (pdir[pid_index][pdir_idx[k]].valid == 0)

      {
        fprintf(fp, "FIRST PAGE FAULT!\n");

        printf("FIRST PAGE FAULT!\n");

        TABLE* table = (TABLE*)calloc(PTLBNUM, sizeof(TABLE));

        pdir[pid_index][pdir_idx[k]].pt = table;
```

```
          pdir[pid_index][pdir_idx[k]].valid = 1;
        }

        TABLE* ptbl = pdir[pid_index][pdir_idx[k]].pt;

        if (ptbl[tbl_idx[k]].valid == 0)

        {
          fprintf(fp, "SECOND PAGE FAULT!\n");

          printf("SECOND PAGE FAULT!\n");

          if (fpl_hd != fpl_tl) {
            ptbl[tbl_idx[k]].pfn = free_page_list[(fpl_hd % FRAMENUM)];

            ptbl[tbl_idx[k]].valid = 1;

            fpl_hd++;

          }

          else {
            for (int k = 0; k < CHILDNUM; k++)

            {
              kill(pid[k], SIGKILL);
            }

            msgctl(msgq, IPC_RMID, NULL);

            exit(0);

            return 0;
          }
        }

        fprintf(
            fp,
            "VA : 0x%08x[PDIR : %d, PTBL : %d, OFFEST : 0x%04x] -> PA:0x%08x\n",
            vm[k], pdir_idx[k], tbl_idx[k], offset[k],
            (ptbl[tbl_idx[k]].pfn << 12) + offset[k]);

        printf(
            "VA : 0x%08x[PDIR : %d, PTBL : %d,OFFSET : 0x%04x] -> PA:0x%08x\n",
            vm[k], pdir_idx[k], tbl_idx[k], offset[k],
            (ptbl[tbl_idx[k]].pfn << 12) + offset[k]);
      }

      memset(&msg, 0, sizeof(msg));
    }
  }

  return 0;
}

void kernelHandler(int sig)

{
  if (proc_done == 1) {
    toFreePageList(pdir[rq[(hd - 1) % QNUM]]);

    proc_done = 0;
  }

  total_tik++;

  count++;

  if (total_tik >= 10000) {
    for (int k = 0; k < CHILDNUM; k++)

    {
      kill(pid[k], SIGKILL);
    }

    msgctl(msgq, IPC_RMID, NULL);

    exit(0);
  }

  fprintf(fp, "------------------------------------------------\n");

  fprintf(fp, "Tik Times : %d ", total_tik);
```

```
  fprintf(fp, "PID :%d ,REMAINING CPU BURST : %d\n", pid[rq[hd % QNUM]],
          cpu_burst[rq[hd % QNUM]]);

  printf("-------------------------------------------------\n");

  printf("Tik Times %d ", total_tik);

  printf("PID : %d ,REMAINING CPU BURST : %d\n", pid[rq[hd % QNUM]],
         cpu_burst[rq[hd % QNUM]]);

  if ((hd % QNUM) != (tl % QNUM)) {
    cpu_burst[rq[hd % QNUM]]--;

    kill(pid[rq[hd % QNUM]], SIGINT);

    if ((count == QUANTUM) | (cpu_burst[rq[hd % QNUM]] == 0)) {
      count = 0;

      if (cpu_burst[rq[hd % QNUM]] != 0) rq[(tl++) % QNUM] = rq[hd % QNUM];

      if (cpu_burst[rq[hd % QNUM]] == 0) {
        cpu_burst[rq[hd % QNUM]] = cpu_burst_ref[rq[hd % QNUM]];

        rq[(tl++) % QNUM] = rq[hd % QNUM];

        proc_done = 1;
      }

      hd++;
    }
  }
}

void processHandler(int sig)

{
  cpu_burst[idx]--;  // 현재 실행중인 프로세스 cpu burst time 감소

  if (cpu_burst[idx] <= 0) cpu_burst[idx] = cpu_burst_ref[idx];

  memset(&msg, 0, sizeof(msg));

  msg.mtype = IPC_NOWAIT;

  msg.pid_index = idx;

  unsigned int temp_add;

  for (int k = 0; k < 10; k++) {
    // 10개의 메모리 접근 시도

    temp_add = (rand() % 1024) << 22;  // 10 bit for directory page table

    temp_add |= (rand() % 1024) << 12;  // 10 bit for page table

    temp_add |= (rand() % 0xfff);  // 12 bit for offset

    msg.vm[k] = temp_add;
  }

  msgret = msgsnd(msgq, &msg, sizeof(msg), IPC_NOWAIT);

  if (msgret == -1) perror("msgsnd error");
}

void toFreePageList(PDIR* p_dir)

{
  PDIR* temp_dir = p_dir;

  for (int i = 0; i < PDIRNUM; i++)

  {
    if (temp_dir[i].valid == 1) {
      temp_dir[i].valid = 0;

      for (int j = 0; j < PTLBNUM; j++)

        if ((temp_dir[i].pt)[j].valid == 1) {
          free_page_list[(fpl_tl++) % FRAMENUM] = (temp_dir[i].pt)[j].pfn;
        }
```

```
      temp_dir[i].pt = NULL;

      free(temp_dir[i].pt);
    }
  }
}
```