시그널 연습문제

□ 문제5 ~ 7

문제 5. 반복문과 슬립을 사용하여 1초 간격으로 '한빛북스' 라고 하는 메시지를 출력하는 프로그램을 작성. 작성된 프로그램은 컨트롤 + C 로 종료되지 않도록 sigprocmask 함수를 이용해서 블록하라.

문제 6. 알람 함수를 이용해서 1초마다 wake up 메시지를 출력하는 프로그램을 작성. SIGALRM 과 핸들러가 실행되는 동안에 SIGINT 를 제외한 모든 시그널을 블록하라.

문제 7. 인터벌 타이머를 이용하여 프로세스의 실행 시간이 1초가 될 때마다 메시지를 출력하는 프로그램을 작성하라.

RAM 은 흔히 두 개의 파트로 구분.
User 메모리와 Kernel 메모리와 I/O 디바이스(물리적 메모리).
read 하게 되면, 커널 메모리에 있는 페이지 캐시가 잡힌 후 데이터가 들어오게 됨. 그리고 이 데이터가 user 에 메모리 버퍼에 들어가게 됨. (copy_to_user)
이렇게 하는 이유는 직접 I/O 디바이스에 접근하게 되면, 아무 데이터나 접근이 가능하게 됨. Protection을 위해 커널로 접근을 제어함. 또한, Prefetch 와 Cacheing 에도 유용. 즉, 많이 쓰는 페이지를 커널 안에 미리 가져옴. 유저가 요청할 때마다 디스크(I/O 디바이스)에서 직접 가져오는 것이 아니라 커널 페이지 안에 있는 내용을 가져옴. 메모리 안에서 데이터를 가져오기 때문에 훨씬 속도가 빠름. 직접, I/O 디바이스에 쓰는 것보다 커널에 쓰고 커널에서 쓸 데이터를 모아서 한 번에 I/O 디바이스에 쓰는 것이 더 좋음.
커널모드로 스위치하여 I/O 디바이스에 있는 디바이스 드라이버를 통해서 데이터를 가져옴. 이것이 유저 버퍼로 들어오게 됨.
단점: copy_to_user, copy_from_user 에서 오버헤드가 발생. (그런데 메모리가 충분하다면 큰 문제가 되지 않음.)
페이지 캐시는 여러 프로세스에 의해 공유되는 페이지임. 만일, 쓸 수 있는 페이지 캐시가 없다면, page replacement 를 해주어야 함.
우선, 페이지 캐시의 내용을 I/O 디바이스 내에 저장한 다음에 요구되는 데이터를 페이지 안으로 가져와야 함.
만일 페이지가 다른 프로세스에 의해 I/O 데이터가 들어오는 중이라면, 바로 교체가 불가능. 이 페이지를 사용하여 수행 중인 I/O 가 끝날 때까지 기다려야 함. 그 다음 replacement 를 하고 수행해야 함.

이 단점은 커버할 수 있는 것이 바로 메모리 매핑임.
I/O 디바이스에서 유저의 페이지로 직접 데이터를 가져오는 것.
I/O 디바이스와 매핑이 되는 유저의 페이지를 잡음. 이를 OS bypass 라고도 함.
직접 가져오므로, read 나 write 를 호출할 필요가 없음. 대신에 map 이나 unmap 과 같은 시스템 호출을 해주어야 함.
이런식으로 데이터를 가져오는 방법에 사용되는 시스템 콜이 메모리 맵(mmap), 언맵(munmap) 임.
단점: 프로세스 모두가 메모리 맵을 사용하려고 한다면, 새로운 프로세스가 사용할 수 있는 메모리 맵이 없을 수 있음. 이는 굉장히 큰 Page Fault 가 발생할 수 있다. 굉장히 많은 Page Fault 가 발생하게 되면 궁극적으로 이를 쓰레싱이라고 함.
그렇기 때문에 위와 메모리 맵을 적절히 조합해서 사용.
DMA (Direct Memory Access), GPU: GPU는 기본적으로 메모리 매핑을 통해 데이터를 가져오게 됨, Kubernetes (docker 기반 플랫폼)

메모리 매핑의 개념

메모리 매핑

- 파일을 프로세스의 메모리에 매핑
- 프로세스에 전달할 데이터를 저장한파일을 직접 프로세스의 가상 주소 공간으로 매핑
- read, write 함수를 사용하지 않고도 프로그램 내부에서 정의한 변수를 사용해 파일에서 데이터를 읽거나 쓸 수 있음

메모리 매핑과 기존 방식의 비교

- 기존 방식

```
fd = open(…);
lseek(fd, offset, whence);
read(fd, buf, len);
```

- 메모리매핑 함수 사용

```
fd = open(…);
addr = mmap((caddr_t)0, len, (PROT_READ|PROT_WRITE), MAP_PRIVATE, fd, offset);
```

페이지 캐시의 주소

시작 주소를 리턴

길이

메모리에 대한 프로텍션

공유가 되는 메모리인가 아닌가

어떤 파일과 맵핑될 것인가.

read 함수를 사용하지 않고도 데이터 접근 가능

메모리 매핑 함수

메모리 매핑: mmap(2)

```
#include <sys/mman.h>                    프로텍션 플래그
void *mmap(void *addr, size_t len, int prot, int flags, int fildes, off_t off);
```

- fildes가 가리키는 파일에서 off로 지정한 오프셋부터 len크기만큼 데이터를 읽어 addr이 가리키는 메모리 공간에 매핑
- prot : 보호모드
  - PROT_READ : 매핑된 파일을 읽기만 함
  - PROT_WRITE : 매핑된 파일에 쓰기 허용
  - PROT_EXEC : 매핑된 파일을 실행가능
  - PROT_NONE : 매핑된 파일에 접근 불가
  - prot에 PROT_WRITE를 지정하려면 flags에 MAP_PRIVATE를 지정하고, 파일을 쓰기 가능 상태로 열어야함 이유: 두 프로세스가 하나의 메모리를 공유하고 있다고 가정하자. 여기서 하나의 프로세스가 메모리를 수정하게 되면 다른 프로세스는 이 수정된 데이터를 원하지 않음. 그래서 이 메모리를 복사하여 새로 씀(Copy on Write)
- flags : 매핑된 데이터를 처리하기 위한 정보 저장
  - MAP_SHARED : 다른 사용자와 데이터의 변경 내용공유
  - MAP_PRIVATE : 데이터의 변경 내용 공유 안함
  - MAP_FIXED : 매핑할 주소를 정확히 지정(권장 안함)
  - MAP_NORESERVE : 매핑된 데이터를 복사해 놓기 위한 스왑영역 할당 안함
  - MAP_ANON : 익명의 메모리 영역 주소를 리턴
  - MAP_ALIGN : 메모리 정렬 지정
  - MAP_TEXT : 매핑된 메모리 영역을 명령을 실행하는 영역으로 사용
  - MAP_INITDATA : 초기 데이터 영역으로 사용

[예제 8-1] mmap 함수 사용하기(1) - test1.c

ex8_1.c

```
...
08  int main(int argc, char *argv[]) {
09      int fd;
10      caddr_t addr;
11      struct stat statbuf;
12
13      if (argc != 2) {
14          fprintf(stderr, "Usage : %s filename\n", argv[0]);
15          exit(1);
16      }
17
18      if (stat(argv[1], &statbuf) == -1) {
19          perror("stat");
20          exit(1);
21      }
22
23      if ((fd = open(argv[1], O_RDWR)) == -1) {
24          perror("open");
25          exit(1);
26      }
27
(다음 쪽)
```

명령행 인자로 매핑할 파일명 입력

[예제 8-1] mmap 함수 사용하기(2)

ex8_1.c

```
28      addr = mmap(NULL, statbuf.st_size, PROT_READ|PROT_WRITE,
29                  MAP_SHARED, fd, (off_t)0);
30      if (addr == MAP_FAILED) {
31          perror("mmap");
32          exit(1);
33      }
34      close(fd);
35
36      printf("%s", addr);
37
38      return 0;
39  }
```

커널에서 알아서 해줌. 메모리 공간을. – 이기현 교수님 –

파일 내용을 메모리에 매핑

매핑한 파일내용 출력

```
[s16010980@sce 1101]$ ls -al > unix.txt
[s16010980@sce 1101]$ ./test1 unix.txt
```

read() 함수를 쓰지 않았음에도 읽어옴.

```
# cat mmap.dat 테스트 데이터의 내용.
HANBIT
BOOK
# ex8_1.out
Usage : ex8_1.out filename
# ex8_1.out mmap.dat 맵핑된 내용이 출력됨.
HANBIT
BOOK
```

메모리 매핑 해제 함수

메모리 매핑 해제: munmap(2)

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

int munmap(void *addr, size_t len);
```

- addr이 가리키는 영역에 len 크기만큼 할당해 매핑한 메모리 해제
- 해제한 메모리에 접근하면 SIGSEGV 또는 SIGBUS 시그널 발생

[예제 8-2] munmap 함수 사용하기 ex8_2.c

```
...
08  int main(int argc, char *argv[]) {
09      int fd;
10      caddr_t addr;
11      struct stat statbuf;
12
13      if (argc != 2) {
14          fprintf(stderr, "Usage : %s filename\n", argv[0]);
15          exit(1);
16      }
17
18      if (stat(argv[1], &statbuf) == -1) {
```

[예제 8-2] munmap 함수 사용하기(2) – test2.c

ex8_2.c

```
19          perror("stat");
20          exit(1);
21  }
22
23      if ((fd = open(argv[1], O_RDWR)) == -1) {
24          perror("open");
25          exit(1);
26      }
27
28      addr = mmap(NULL, statbuf.st_size, PROT_READ|PROT_WRITE,
29                  MAP_SHARED, fd, (off_t)0);
30      if (addr == MAP_FAILED) {
31          perror("mmap");
32          exit(1);
33      }
34      close(fd);
35
36      printf("%s", addr);
37
38      if (munmap(addr, statbuf.st_size) == -1) {
39          perror("munmap");
40          exit(1);
41      }
42
43      printf("%s", addr);
44
45      return 0;
46  }
```

파일 내용을 메모리에 매핑

메모리 매핑 해제

매핑이 해제된 메모리에 접근

```
# ex8_2.out mmap.dat
HANBIT
BOOK
세그멘테이션 결함(Segmentation Fault)(코어 덤프)
```

메모리 매핑의 보호모드 변경

보호모드 변경: mprotect(2)

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

int mprotect(void *addr, size_t len, int prot);
```

- mmap 함수로 메모리 매핑을 수행할 때 초기값을 설정한 보호모드를 mprotect 함수로 변경 가능
- prot에 지정한 보호모드로 변경
  - PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE

파일의 크기 확장 함수

파일의 크기와 메모리 매핑

- 존재하지 않거나 크기가 0인 파일은 메모리 매핑할 수 없음
- 빈 파일 생성시 파일의 크기를 확장한 후 메모리 매핑을 해야함

경로명을 사용한 파일 크기 확장: truncate(3)

```
#include <unistd.h>

int truncate(const char *path, off_t length);
```

- path에 지정한 파일의 크기를 length로 지정한 크기로 변경

파일 기술자를 사용한 파일 크기 확장: ftruncate(3)

```
#include <unistd.h>

int ftruncate(int fildes, off_t length);
```

- 일반 파일과 공유메모리에만 사용가능
- 이 함수로 디렉토리에 접근하거나 쓰기 권한이 없는 파일에 접근하면 오류 발생

디렉토리는 확장 불가.

[예제 8-3] ftruncate 함수 사용하기(1) – test3.c

ex8_3.c

```
...
09  int main(void) {
10      int fd, pagesize, length;
11      caddr_t addr;
12
13      pagesize = sysconf(_SC_PAGESIZE);
14      length = 1 * pagesize;
15
16      if ((fd = open("m.dat", O_RDWR | O_CREAT | O_TRUNC, 0666))
== -1) {
17          perror("open");
18          exit(1);
19      }
20
21      if (ftruncate(fd, (off_t) length) == -1) {
22          perror("ftruncate");
23          exit(1);
24      }
25
```

우선, 페이지 사이즈를 찾음.
일단, 우리는 4 Kbytes.

메모리의 페이지 크기정보 검색

빈 파일의 크기 증가

[예제 8-3] ftruncate 함수 사용하기(2)

ex8_3.c

```
26      addr = mmap(NULL, length, PROT_READ|PROT_WRITE, MAP_SHARED,
                    fd, (off_t)0);
27      if (addr == MAP_FAILED) {
28          perror("mmap");
29          exit(1);
30      }
31
32      close(fd);
33
34      strcpy(addr, "Ftruncate Test\n");
35
36      return 0;
37  }
```

메모리 매핑

매핑한 메모리에 데이터 쓰기

```
# ls m.dat
m.dat: 해당 파일이나 디렉토리가 없음
# ex8_3.out
# cat m.dat
ftruncate Test
```

매핑된 메모리 동기화

☐ 매핑된 메모리 동기화

- 매핑된 메모리의 내용과 백업 내용이 일치하도록 동기화 필요
  매핑된 메모리의 내용이 바로 I/O 디바이스로 가지는 않음. 일반적으로는 주기적으로 30초에 한번씩 메모리에 있는 내용이 I/O 디바이스에 저장됨. 중요한 데이터라면, 그 30초 사이에 I/O 디바이스에 저장하고자 한다면, 동기화 필요.

☐ 매핑된 메모리 동기화: msync(3)

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

int msync(void *addr, size_t len, int flags);
```

- addr로 시작하는 메모리 영역에서 len 길이만큼의 내용을 백업저장장치에 기록
- flags : 함수의 동작 지시
  - MS_ASYNC : 비동기 쓰기 작업
  - MS_SYNC : 쓰기 작업을 완료할 때까지 msync 함수는 리턴 안함 안전하지만, ASYNC 보다는 느림.
  - MS_INVALIDATE : 메모리에 복사되어 있는 내용을 무효화

[예제 8-4] msync 함수 사용하기(1) – test4.c

ex8_4.c

```
...
08  int main(int argc, char *argv[]) {
09      int fd;
10      caddr_t addr;
11      struct stat statbuf;
12
13      if (argc != 2) {
14          fprintf(stderr, "Usage : %s filename\n", argv[0]);
15          exit(1);
16      }
17
18      if (stat(argv[1], &statbuf) == -1) {
19          perror("stat");
20          exit(1);
21      }
22
23      if ((fd = open(argv[1], O_RDWR)) == -1) {
24          perror("open");
25          exit(1);
26      }
```

파일의 상세 정보 검색

[예제 8-4] msync 함수 사용하기(2)

ex8_4.c

```
28      addr = mmap(NULL, statbuf.st_size, PROT_READ|PROT_WRITE,
29                  MAP_SHARED, fd, (off_t)0);
30      if (addr == MAP_FAILED) {
31          perror("mmap");
32          exit(1);
33      }
34      close(fd);
35
36      printf("%s", addr);
37
38      printf("---------\n");
39      addr[0] = 'D';
40      printf("%s", addr);
41
42      msync(addr, statbuf.st_size, MS_SYNC);
43
44      return 0;
45  }
```

메모리 매핑

매핑된 내용 출력

MS_ASYNC 는 동기화를 하지만, 다른 프로세스를 동시에 수행.

매핑된 내용 수정

끝날 때까지 리턴하지 않음.

수정된 내용 동기화

```
# cat mmap.dat
HANBIT
BOOK
# ex8_4.out mmap.dat
HANBIT
BOOK
---------
DANBIT
BOOK
# cat mmap.dat
DANBIT
BOOK
```

I/O 디바이스에 있는 수정된 데이터가 출력되는 것.

MAP_PRIVATE 으로 한 다음에 msync 를 하지 않으면, 실제 데이터는 변경되지 않음.

실습할 때는 되도록 MAP_SHARED 로 하도록 함.

[예제 8-5] 데이터 교환하기(1) – test5.c

ex8_5.c

메모리 매핑을 이용한 데이터 교환

- 부모 프로세스와 자식 프로세스가 메모리 매핑을 사용하여 데이터 교환 가능

```
...
09  int main(int argc, char *argv[]) {
10      int fd;
11      pid_t pid;
12      caddr_t addr;
13      struct stat statbuf;
14
15      if (argc != 2) {
16          fprintf(stderr, "Usage : %s filename\n", argv[0]);
17          exit(1);
18      }
19
20      if (stat(argv[1], &statbuf) == -1) {
21          perror("stat");
22          exit(1);
23      }
24
```

[예제 8-5] 데이터 교환하기(2)

ex8_5.c

```
25      if ((fd = open(argv[1], O_RDWR)) == -1) {
26          perror("open");
27          exit(1);
28      }
29
30      addr = mmap(NULL, statbuf.st_size, PROT_READ|PROT_WRITE,
31      MAP_SHARED, fd, (off_t)0);
32      if (addr == MAP_FAILED) {
33          perror("mmap");
34          exit(1);
35      }
36      close(fd);
37
38      switch (pid = fork()) {
39          case -1 :  /* fork failed */
40              perror("fork");
41              exit(1);
42              break;
```

부모와 자식 프로세스가 암묵적으로 별도의 페이지를 사용하여 공유함.

부모와 자식이 공유하므로,

메모리 매핑

fork 함수로 자식 프로세스 생성

[예제 8-5] 데이터 교환하기(3)

ex8_5.c

```
43          case 0 :   /* child process */
44              printf("1. Child Process : addr=%s", addr);
45              sleep(1);
46              addr[0] = 'x';
47              printf("2. Child Process : addr=%s", addr);
48              sleep(2);
49              printf("3. Child Process : addr=%s", addr);
50              break;
51          default :  /* parent process */
52              printf("1. Parent process : addr=%s", addr);
53              sleep(2);
54              printf("2. Parent process : addr=%s", addr);
55              addr[1] = 'y';
56              printf("3. Parent process : addr=%s", addr);
57              break;
58      }
59
60      return 0;
61  }
```

슬립을 통해 동기화 수행(안전한 방법이 아님.)

자식 프로세스가 매핑된 내용 수정

자식 프로세스가 깨어나 부모 프로세스가 수정한 내용을 읽음

부모 프로세스가 매핑된 내용 수정

부모 프로세스가 깨어나 내용 수정.

```
[s16010980@sce 1101]$ vi test5.c
[s16010980@sce 1101]$ vi testdata
[s16010980@sce 1101]$ gcc -o test5 test5.c
[s16010980@sce 1101]$ ./test5 testdata
1. Parent process : addr=This is UNIX Class
1. Child Process : addr=This is UNIX Class
2. Child Process : addr=xhis is UNIX Class
2. Parent process : addr=xhis is UNIX Class
3. Parent process : addr=xyis is UNIX Class
[s16010980@sce 1101]$ 3. Child Process : addr=xyis is UNIX Class
```

```
# cat mmap.dat
HANBIT BOOK
# ex8_5.out mmap.dat
1. Child Process : addr=HANBIT BOOK
1. Parent process : addr=HANBIT BOOK
2. Child Process : addr=xANBIT BOOK
2. Parent process : addr=xANBIT BOOK
3. Parent process : addr=xyNBIT BOOK
3. Child Process : addr=xyNBIT BOOK
# cat mmap.dat
xyNBIT BOOK
#
```