Chungnam National University
Department of Computer Science and Engineering

2012년 가을학기

기말고사 모범답안

2012년 12월 14일
시스템 프로그래밍

분반/학번	반
이름	

문제	배점	점수
1	20	
2	20	
3	20	
4	20	
5	20	
6	10	
총계	110	

나는 이 답안을 부정행위 없이, 내 스스로의 힘으로 작성하였으며, 다른 학생의 것을 보거나, 다른 학생에게 보여주지 않았음을 맹세합니다. ( )

문제 1. 기초지식 (20점)

1) (4점) 좀비 프로세스란 무엇이며, 좀비가 많으면 무슨 문제가 생기는가?

- 프로세스는 종료되었지만, 시스템 자원을 사용하는 프로세스
- 부모 프로세스가 아직 정리해 주지 않을 때 발생한다.
- 시스템의 메모리가 부족해진다. 메모리 누수가 발생한다

– 설명이 맞으면 2점, 문제가 맞으면 2점

2) (4점) TLB는 무엇이며, 무슨 장점이 있는가 ?

– 페이지 테이블을 저장하는 캐시의 역할로 MMU에 들어 있음

– 페이지 테이블 접근을 위한 메모리 접근을 없애주고, 빠른 캐시로 페이지 테이블을 가져올 수 있다

– 설명이 맞으면 2점, 장점이 맞으면 2점

3) (4점) 가상메모리의 세 가지 주요 기능을 쓰시오.

– 가상주소 공간에 대한 DRAM 캐시의 역할

– 메모리 관리의 도구/수단으로 사용

– 메모리 보호 방법으로 사용

– 다 맞으면 4점, 한개당 1점씩

4) (4점) 동기형 예외에는 Trap, Faults, Abort의 세 가지 종류가 있다. 이 중에서 시스템 콜과 Breakpoint 등을 구현하기 위해 사용되는 것은 무엇인가?

– trap

5) (4점) 내부단편화(Internal fragmentation)가 생기는 원인에 대해 설명하시오.

– 힙 메모리 자료구조의 유지, 메모리 정렬요건

– 한 개만 서도 4점

문제 2. (20점) [시그널] 문제 2. (20점) [시그널] 아래 프로그램은 자식프로세스를 생성하여 date 란 프로그램을 지속적으로 실행시키는 프로그램이다. 부모 프로세스는 자신의 자식 프로세스들을 joblist 로 관리하며, 하나의 자식 프로세스에 대해 한 개의 job 엔트리를 추가한다. addjob()과 deletejob()은 이 joblist로부터 엔트리를 추가하거나 삭제한다. 부모프로세스는 실행된 프로그램의 job 들을 관리하고 자식이 종료되었을 경우 자식을 청소하는 코드로 구성되어 있다.

```
void handler(int sig){
    pid_t pid;
    while ((pid = waitpid(-1, NULL, 0)) > 0)
        deletejob(pid);
}
int main(int argc, char **argv)
{
    int pid;
    initjobs();

   /* (A) SIGCHILD 핸들러 등록 */
    while (1) {
        if ((pid = Fork()) == 0) {
            Execve("/bin/date", argv, NULL);
        }
        addjob(pid);
    }
    exit(0);
}
```

1) (4점)위 프로그램에서 SIGCHILD 핸들러를 등록하기 위해 (A)에 들어가야 하는 프로그램 문장을쓰시오.

- signal(SIGCHILD, handler)
정확하게 쓰지 않았다면 틀림

2) (6점) 여러분이 이미 실습시간에 경험했듯이 위 프로그램을 실행시키면 미묘한 오류를 발생시킬 수 있는데, 이 오류가 발생하게 되는 프로그램 실행 상황을 addjob( )과 deletejob( )을 이용해서 설명하시오.

- 만일 자식 프로세스가 먼저 진행하게 되면, deletejob( )이 addjob( )보다 먼저실행되는 경우에는 joblist에 없는 프로세스를 삭제하려 시도하고, 결국 이 프로세스는 영원히 삭제할 수 없게 된다. 이러한 현상을 data race라고 부른다.
- 부분점수 가능

3) (4점) 위 프로그램이 정상적으로 실행되도록 하기 위해 01반의 이XXX 학생이 다음과 같이 코드를 수정하였다. 이 학생이 사용한 sigprocmask 함수의 역할은 무엇인지 설명하라.

- 특정 시그널의 수신을 블록 또는 해제해준다
 시그널 블록/해제가 정확하게 들어가야 하며 다른 설명은 틀림

```
int main(int argc, char **argv)
{
    int pid;
    sigset_t mask;
    /*  (A) SIGCHILD 핸들러 등록 */
    initjobs();

    while (1) {
          Sigemptyset(&mask);
          Sigaddset(&mask, SIGCHLD);
          Sigprocmask(SIG_BLOCK, &mask, NULL);
          if ((pid = Fork()) == 0) {
              Sigprocmask(SIG_UNBLOCK, &mask, NULL);
              Execve("/bin/date", argv, NULL);
          }
          Sigprocmask(SIG_UNBLOCK, &mask, NULL);
          addjob(pid);
    }
    exit(0);
}
```

4) (6점) 3)번에 작성된 프로그램은 여전히 정상동작하지 않는다. 이 프로그램의 무엇이 문제인지 지적하고, 코드를 수정하라.

- 부모가 addjob( ) 한 후에 시그널을 UNBLOCK 해야 한다

```
while (1) {
          Sigemptyset(&mask);
          Sigaddset(&mask, SIGCHLD);
          Sigprocmask(SIG_BLOCK, &mask, NULL);
          if ((pid = Fork()) == 0) {
              Sigprocmask(SIG_UNBLOCK, &mask, NULL);
              Execve("/bin/date", argv, NULL);
          }
          addjob(pid);
          Sigprocmask(SIG_UNBLOCK, &mask, NULL);
    }
    exit(0);
```

문제 3. (20점) **[동적메모리 할당]**

1) (4점) 다음과 같은 순서로 malloc을 요청하는 경우에 블록의 크기와 헤더에 저장되는 값을 쓰시오. 헤더는 4바이트를 사용하며, 할당시에는 double-alignment로 8의 배수크기로 할당하며, 헤더만 있는 간접리스트 방식을 사용한다고 가정한다.

- 각 1점씩

요청	블록크기(바이트)	블록 헤더
malloc(12)	4+12=16	0x11
malloc(13)	4+13+7=24	0x19

2) (4점) 간접리스트(Implicit) 방식에서 Free 메모리 블럭을 찾아서 할당하려고 할 때 First Fit, Next Fit, Best Fit 의 세 가지 방식을 고려하고 있다. 이 중에서 Utilization 성능지표를 극대화 할 수 있는 방법은 어느 것인가 ?

- Best Fit

3) (4점) 양방향 포인터를 사용하는 직접리스트(Explicit list) 방식으로 malloc( )을 구현할 때 free블럭을 검색하는 속도를 간접리스트(Implicit list) 방식에서와 비교할 때 어느 쪽이 더 빠른지 설명하고, 그 이유를 설명하시오.

- 직접리스트가 더 빠르다. 그 이유는 직접리스트에서는 free한 블록만 관리하지만, 간접리스트는 모든 블록을 관리해야 하기 때문이다

일부 다른 이유를 적은 경우 감점 있음

- 하나만 맞으면 2점, 다 맞으면 4점

4) (4점) 메모리 블록을 할당할 때, 요청한 데이터보다 더 큰 사이즈의 블록을 할당하는 대신, 이 블록을 둘로 나누고, 새로운 free블록을 만드는 것이 메모리 이용도(Utilization) 측면에서 유리해진다. 그 이유를 단편화와 연관지어 설명하시오.

- 블록을 나누지 않았을 경우 내부 단편화가 될 수 있음을 정확하게 설명해야 함.

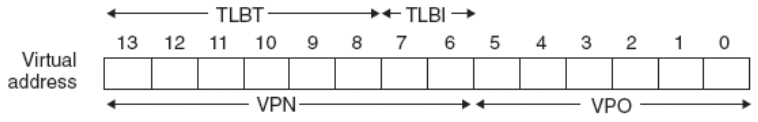
일부 설명이 맞으면 부분점수 또는 틀린 설명의 경우 감점있음

5) (4점) Segregated list(분리 리스트) 방식을 사용하면 무슨 장점이 생기는지 설명하시오.

-가용블록 검색 속도가 빨라진다, 단편화가 줄어든다.

분리 리스트의 특성을 이해하고(사이즈별로 free 블록을 관리하기 때문에) 처리율과 사용률 모두 좋아짐. 이유 분리 리스트 특성 또는 이유 없음 -2 감점, 처리율, 사용율 각 1점

문제 4. 가상메모리 (20점) 어떤 컴퓨터가 14비트의 가상주소, 12비트의 물리주소를 가지며, 어느 한 시점에 TLB와 캐시, 페이지테이블의 내용이 아래와 같다고 하자. TLB와 캐시의 접근 방식은 아래 그림과 같다.

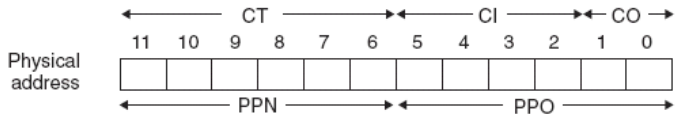


Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

(a) TLB: Four sets, 16 entries, four-way set associative

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	—	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	–	0
04	—	0	0C	–	0
05	16	1	0D	2D	1
06	—	0	0E	11	1
07	—	0	0F	0D	1

(b) Page table: Only the first 16 PTEs are shown

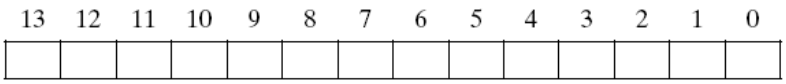


Idx	Tag	Valid	Blk 0	Blk 1	Blk 2	Blk 3
0	19	1	99	11	23	11
1	15	0	—	—	—	—
2	1B	1	00	02	04	08
3	36	0	—	—	—	—
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	—	—	—	—
7	16	1	11	C2	DF	03
8	24	1	3A	00	51	89
9	2D	0	—	—	—	—
A	2D	1	93	15	DA	3B
B	0B	0	—	—	—	—
C	12	0	—	—	—	—
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	—	—	—	—

(c) Cache: Sixteen sets, 4-byte blocks, direct mapped

가상주소 0x027c 를 CPU가 접근하려고 할 때, 메모리로 부터 어떤 값을 읽어오게 되는지의 주소번역과정에 따라 아래 빈 곳을 채우시오.

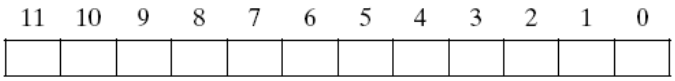
A. Virtual address format



B. Address translation

Parameter	Value
VPN	
TLB index	
TLB tag	
TLB hit? (Y/N)	
Page fault? (Y/N)	
PPN	

C. Physical address format



D. Physical memory reference

Parameter	Value
Byte offset	
Cache index	
Cache tag	
Cache hit? (Y/N)	
Cache byte returned	

A. 00 0010 0111 1100

B.

VPN:	0x9
TLBI:	0x1
TLBT:	0x2
TLB hit?	N
page fault?	N
PPN:	0x17

C. 0101 1111 1100

D.

CO:	0x0
CI:	0xf
CT:	0x17
cache hit?	N
cache byte?	-

D, Cache byte : MEM, DISK −, 빈칸 도 맞음
단 D 를 제대로 풀지 못하고 빈칸을 둔 경우에는 틀림

− 채점 방법

A : 2점

B : VPN : 1점
TLBI : 1점
TLBT : 1점
TLB Hit : 2점
Page fault : 2점
PPN : 2점

C : 2점

D :
CO : 1점
CI : 1점
CT : 1점
cache hit : 2점
cache byte : 2점

문제 5. 문제 5. (20점) 동적메모리 할당] 다음의 코드는 header 와 footer 를 사용하는 간접리스트방식에서의 dynamic memory allocator 의 일부이다.

```
#define WSIZE       4
#define DSIZE       8
#define PACK(size, alloc)  ((size) | (alloc))
#define GET(p)       (*(unsigned int *)(p))
#define PUT(p, val)  (*(unsigned int *)(p) = (val))
#define GET_SIZE(p)  (GET(p) & ~0x7)
#define GET_ALLOC(p) (GET(p) & 0x1)
#define HDRP(bp)       ((char *)(bp) - WSIZE)
#define FTRP(bp)       ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)
#define NEXT_BLKP(bp)  ((char *)(bp) + GET_SIZE(((char *)(bp) - WSIZE)))
#define PREV_BLKP(bp)  ((char *)(bp) - GET_SIZE(((char *)(bp) - DSIZE)))
static char *heap_listp = 0;

void mm_free(void *bp)
{
    size_t size = GET_SIZE(HDRP(bp));
    PUT(HDRP(bp), PACK(size, 0));
    PUT(FTRP(bp), PACK(size, 0));
}


static void *func(void *bp) {
    size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));

    /* Case 1 */
    if ( ①___________________________________) {
        return bp;
    }
    /* Case 2 */
    else if (②_____________________________________) {
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(bp), PACK(size, 0));
        PUT(FTRP(bp), PACK(size,0));
    }
    /* Case 3 */
    else if (③_____________________________________) {
        size += GET_SIZE(HDRP(PREV_BLKP(bp)));
        PUT(FTRP(bp), PACK(size, 0));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }
    /* Case 4 */
    else if(④_____________________________________){
        size += GET_SIZE(HDRP(PREV_BLKP(bp))) +  GET_SIZE(FTRP(NEXT_BLKP(bp)));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }
return bp; }
```

1) (5 점) 위 코드 중 mm_free 함수는 dynamic memory allocator 가 가져야 할 성능지표인 처리량(Throughput), 사용률(Utilization) 중 하나에 문제가 있을 수 있다. 어떤 지표가 안 좋을 수 있는가? (나머지 함수와 코드는 모두 문제가 없다고 가정한다.)

- 사용률

2) (5 점) 이처럼 단순한 mm_free 함수를 제공하면 어떤 문제가 발생할 수 있는지 단편화와 연관하여 그 이유를 밝히고 해결책을 논하시오

1. - 결합(coalese) 과정이 없기 때문에 external fragmentation 이 발생할 수 있으므로 메모리 사용율이 안 좋아진다.(2 점) => external fragmentation 에 대한 내용을 꼭 써야 하며 이와 동일한 의미를 적으면 정답
2. 해결책 => free 후에 결합과정을 더한다. => 결합과정이 없기 때문에 들어가야 한다 등등 에 대한 의미일 경우 (2 점), 직접결합, 지연결합 상관없이 external fragmentation 을 줄이기 위한 노력에 대한 내용이면 됨

다 맞으면 5점

3) (5 점)위에서 밝힌 해결책을 바탕으로 새롭게 구현한 func 함수내의 ①,②,③,④ 조건을 완성하시오

- prev_alloc && next_alloc
- prev_alloc && !next_alloc
- !prev_alloc && next_alloc
- !prev_alloc && !next_alloc

동작하는 코드를 정확히 작성했으면 정답. 1,2,3,4, 순서가 틀리면 틀림

부분점수 각 1점

4) (5 점) 2)에서 제시한 free 함수 성능 개선 방법을 적용하면 이번에는 반대의 지표가 나빠질 수 있다. 이를 보완할 수 있는 방법을 한 가지만 제안하라.

Free 후 결합을 해야 하기 때문에 반대로 throughput 이 안 좋아질 수 있다. 따라서 매번 free 할 때 결합할 것이 아니라 일정 threshold 값이 넘을 경우에만 결합을 시도한다.

지연통합이 아니더라도 throughput 이 좋아질 수 있는 방법을 제시하면 (5점)

- 반드시 2번에서 제안한 방식과 연계된 개선방법을 제시해야 하며, 이와 무관한 일반적인 성능 개선 방법을 제시하면 안됨

Free 에서 결합 시 사용율은 올라가지만 처리율은 떨어질 수 있다는 사실을 정확히 이해하고 있어야 하며, 결합시 발생한 처리율에 관련된 오버헤드를 줄일 수 있는 방법을 제안 해야 함. 일부 부분점수 및 감점 있음

문제 6. 기타 (10점) 다음과 같은 프로그램을 보고 질문에 답하시오.

```
int main () {
    if (fork() == 0) {
        if (fork() == 0) {
            printf("3");
        }
        else {
            pid_t pid; int status;
            if ((pid = wait(&status)) > 0) {
                printf("4");
            }
        }
    }
    else {
        printf("2");
        exit(0);
    }
    printf("0");
    return 0;
}
```

아래의 5개의 출력에 대해서 이 프로그램으로부터 출력될 수 있는지, 없는지 결정하시오.

(A) 32040 Y (1) 있다 (2) 없다
(B) 34002 N (1) 있다 (2) 없다
(C) 30402 Y (1) 있다 (2) 없다
(D) 23040 Y (1) 있다 (2) 없다
(E) 40302 N (1) 있다 (2) 없다