Chungnam National University
Department of Computer Science and Engineering

2014년 가을학기

기말고사 모범답안

2014년 12월 11일
시스템 프로그래밍

분반/학번	반 번
이름	

문제	배점	점수
1	20	
2	14	
3	20	
4	36	
5	20	
총계	110	

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

문제 1. 기초지식 (20점)

1) (4점) 시스템 콜 이란 무엇인지 설명하시오.

운영체제가 응용프로그램에게 제공하는 서비스들의 API들.

2) (4점) 운영체제가 다수의 프로세스들을 멀티태스킹 형태로 실행시켜줄 때 문맥전환 (Context Switch) 라는 서비스를 해준다. 문맥전환에서 해주는 일이 무엇인지 설명하시오.

현재 실행중인 프로세스들이 사용하던 레지스터 값, 프로세스 관련 자료구조들을 메모리에 저장하고, 새롭게 실행해야 하는 프로세스들이 사용하던 레지스터 값, 프로세스 자료구조를 메모리로부터 읽어와서 레지스터에 써주고, 자료구조를 사용할 수 있도록 해주는 작업.

3) (4점) 프로세스가 종료되면 좀비가 된다. 좀비들을 제거하지 않으면 메모리 누수 (memory leak)이 발생한다. 메모리 누수는 웹서버나 쉘 프로그램과 같은 경우에 더욱 심각한 문제가 될 수 있는데, 이들 프로그램에서 더 심각한 문제가 되는 이유를 설명하시오.

웹서버나 쉘 프로그램들은 자신들이 죽지 않고 오랜기간 실행되는 프로그램들로 init 프로세스가 좀비를 제거하지 못하므로, 웹서버나 쉘 프로그램등이 직접 좀비를 제거하지 않으면 시스템이 다운될 수 있음

4) (4점) 동적메모리 할당 방법중에서 버디시스템(Buddy System)에 대해서 설명하시오.

Segregated 메모리 할당방식의 하나로, 각 크기 클래스가 2의 제곱인 특수한 경우의 격리 메모리 할당방식

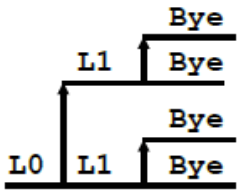
5) (4점) 컴퓨터시스템에서 어떤 프로그램이 실행되는 중간에 예외적인 사건들, 예를 들어 인터럽트가 발생한 경우, 인터럽트를 처리하는 과정을 순서가 드러나게 설명하시오.

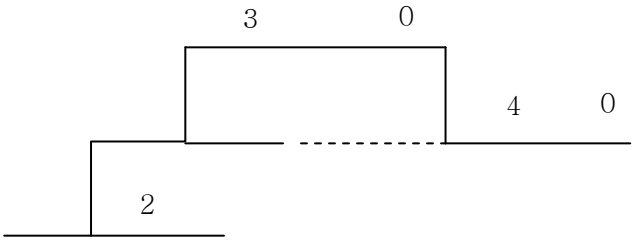
현재 실행중이던 프로그램을 중지하고, 인터럽트 핸들러로 이동.
인터럽트 핸들러에서 인터럽트 요청을 서비스해주고, 핸들러 종료.
핸들러 종료후, 이전에 중지되었던 프로그램의 실행을 재개

문제 2. (14점) [프로세스의 제어] 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;
}
```

1) (4점) 위 프로그램의 실행결과를 아래 예제처럼 프로세스 그래프로 나타내시오. 그래프에는 출력되는 결과를 표시해야 한다.





예)

2) (10점) 아래의 (A)~(E)의 5개의 출력이 위 프로그램으로부터 출력될 수 있는지, 없는지 판단하시오.(각2점)

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

문제 3. (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)  // (A)
        deletejob(pid);
}
int main(int argc, char **argv)
{
    int pid;

    signal(SIGCHLD, handler) ;
    initjobs();

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

1) (4점)위 프로그램의 (A) 부분에서 종료된 자식 프로세스의 처리를 위해 while 루프형태로 구현해야 하는 이유를 설명하시오.

연속된 시그널이 발생하더라도 핸들러에서 최대한 많이 동일 시그널을 처리하려고. 동일 시그널이 연속 발생하면 이것을 큐에 저장해주지 않기 때문에 해줘야 하는 일

2) (6점) 여러분이 이미 실습시간에 경험했듯이 위 프로그램을 실행시키면 동시에 실행되는 두개의 프로세스들(concurrent processes) 때문에 race condition(경쟁상태) 라는 미묘한 오류를 발생시킬 수 있는데, 이 오류가 발생하게 되는 실행과정을 addjob( )과 deletejob( )을 이용해서 설명하시오.

자식프로세스를 fork&exec한 후에 생성된 자식 프로세스를 addjob 하는 동안에 자식이 먼저 종료되어 SIGCHILD가 발생하면, 시그널 핸들러에서 addjob이 미처 되어 있지 않는 프로세스를 deletejob 하려다가 에러가 난다

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

```
int main(int argc, char **argv)
{
    int pid;
    sigset_t mask;

    initjobs();
    signal(SIGCHLD, handler);
    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);
}
```

Mask 에 지정된 시그널의 처리를 막거나, 처리가 금지된 시그널의 처리를 가능하게 한다

4) (6점) 위 3)번에 작성된 프로그램에 의해 어떻게 race condition상태를 회피할 수 있는지, 설명하시오. 구체적이고 정확한 설명을 해야함. 2)번에서 제시한 문제 상태가 3)번 코드에 의해 어떻게 회피될 수 있는지 설명해야 함

위 3)번의 코드에서 부모는 sigprocmask를 이용하여 SIGCHILD의 처리를 정지시킨 후에 자식을 생성한다. 자식이 생성된 후, 자식은 부모의 시그널 관련 자료구조들(문맥데이터)을 모두 복사해 가지고 있으므로, SIGCHILD가 block 되어 있을 것이므로, 자식쪽에서는 SIGCHILD를 일단 해제해주고 난 후에 exec을 실행해준다. 자식이 exec으로 실행후에 종료가 되어 SIGCHILD가 부모에게 날라가지만, 부모는 여전히 SIGCHILD가 블록된 상태이므로, 유유히 자신의 코드를 실행해서 addjob을 실행해주고 그 뒤에 sigprocmask를 실행해서 막아놨던 시그널을 해제해준다. 이렇게 하면, 자식이 먼저 실행되어서 deltejob이 먼저 실행되는 경우는 발생하지 않는다. 반대로 부모가 먼저 실행된다면, 당연히 문제가 없이 실행되므로 역시 문제 없게 된다.

문제 4. (36점) **[동적메모리 할당]** 다음과 같은 정렬 요건과 블록구조 조합을 이용하는 메모리 할당기를 **64비트 I7 프로세서를** 사용하는 컴퓨터에서 구현하려고 한다. 64비트 프로세서에서는 주소를 64비트로(8바이트) 표시한다. 직접가용리스트(Explicit list)를 사용하며, 가용블록 내에는 pred와 succ 포인터를 포함하며, 할당블록의 경우 데이터(payload)의 크기는 0바이트는 허용하지 않는다. 그러나, free블록의 경우는 데이터 크기가 0인 경우도 가능하다. 헤더와 풋터는 4바이트씩 사용한다. 이전블록과 다음가용블록의 주소를 저장하는 pred와 succ 포인터는 각각 8바이트씩 할당한다.

구현방법	정렬규칙	할당블록	가용블록
1	Word(8바이트)	Header and footer	Header and footer Pred, succ
2	Double words(16바이트)	Header, no footer	Header and footer Pred, succ

1) (4점) 1번 구조를 이용하는 경우, 할당된 블록의 최소 크기와 가용블록(free block)의 최소 크기는 각각 몇 바이트인가? 계산 과정을 보여야 한다.

a) 할당된 블록들의 최소 크기 : 헤더+풋터+데이터 1바이트 = 9, 8바이트 정렬이므로, 16바이트 필요

b) 가용 블록의 최소크기 : 헤더+풋터+pred+succ = 4+4+8+8 = 24, 이미 8의 배수이므로 24바이트 필요

2) (4점) 1)번 문제의 결과로부터 1번 구조를 이용하는 메모리 할당 라이브러리를 구현하려면, 최소 블록의 크기는 몇 바이트가 되어야 하는가?

a), b) 중 큰 크기가 최소블록 크기가 됨. 따라서 24바이트

3) (4점) 2번 구조를 이용하는 경우, 할당된 블록의 최소 크기와 가용블록의 최소 크기는 각각 몇 바이트인가? 계산 과정을 보여야 한다.

a) 할당된 블록들의 최소 크기 : 헤더+풋터+1 = 4+4+1 = 9, 16바이트 정렬이므로, 16바이트가 최소블럭

b) 가용 블록의 최소크기 : 헤더+풋터+pred+succ = 4+4+8+8 = 24바이트. 16바이트 정렬이므로 16의 배수로 맞춰야 하므로 32바이트 필요

4) (4점) 3)번 문제의 결과로부터 2번 구조를 이용하는 메모리 할당 라이브러리를 구현하려면, 최소 블록의 크기는 몇 바이트가 되어야 하는가?

32바이트
부분점수 없음

5) (4점) 위에서 계산해 본 것처럼, 어떤 동적메모리 할당기의 구현에서 최소블록의 크기가 크게 되면, 작은 경우보다 단편화(fragmentation)의 관점에서 어떤 단점이 생기는가?

내부단편화가 커져서 메모리 효율이 낮아진다
부분점수 가능

6) (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 빠름. 이유는 직접 리스트는 free블록들만 검색하면 되지만, 간접리스트는 모든 블록을 검색해야 하므로
부분점수 가능

7) (4점) 할당되었던 메모리 블록을 반환할 때, 해당 크기 클래스의 가용리스트의 제일 앞에 반환하는 방법을 LIFO 방식이라고 부른다. LIFO방식의 반환 알고리즘 구현 시 처리속도(throughput)과 메모리 효율(utilization) 은 각각 어떤 성능을 보이는지 설명하시오.

처리속도는 빨라지고, 메모리 효율은 나빠진다

8) (4점) 구분가용리스트(Segretated list) 방식의 처리속도와 메모리 효율을 직접리스트를 사용하는 경우와 비교하여 장점과 단점을 설명하시오.

가용블록 검색속도는 빨라지고, 메모리 효율도 좋아진다. 그러나, 크기 클래스를 유지하기 위한 관리작업이 늘어난다.

9) (4점) malloc 함수를 구현할 때, 필요한 크기만큼만 할당하고, 기존 free블록의 남는 부분을 쪼개서 새로운 free블록을 추가하는 방법을 사용하면, 좋아지는 것은 무엇이고, 나빠지는 것은 무엇인지 쓰시오.

메모리 효율은 좋아지고, 처리속도는 나빠진다
부분점수 가능

문제 5. (20점) [동적메모리 할당] 다음의 코드는 header 와 footer 를 사용하는 간접리스트방식에서의 동적메모리 할당기 라이브러리 코드의 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) (4 점) 위 코드의 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.)

메모리 효율이 나빠진다.

2) (4 점) 이처럼 단순한 mm_free 함수를 제공하면 어떤 문제가 발생할 수 있는지 단편화와 연관하여 그 이유를 밝히고 해결책을 논하시오

인접한 free블록들이 연결되지 않아서 외부 단편화가 늘어날 수 있다.
Coalescing, 즉 블록연결 과정이 필요한 이유를 설명하면 점수를 줌

3) (8 점)위에서 밝힌 해결책을 바탕으로 새롭게 구현한 func 함수내의 ①,②,③,④ 조건을 완성하시오.

블록 연결의 네가지 경우를 구현하는 것임 각각 2 점씩.

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

4) (4 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 (4점)
- 반드시 2번에서 제안한 방식과 연계된 개선방법을 제시해야 하며, 이와 무관한 일반적인 성능 개선 방법을 제시하면 안됨