Department of Computer Science and Engineering
Chungnam National University

10/29/2009

Unix System Programming 답안 **Hyungshin Kim**

1. [기본] (20점) 다음을 간단히 답하시오.

1) (5점) 리눅스 또는 유닉스의 특징을 2가지 이상 쓰시오.

멀티태스킹 지원, 강력한 네트워크 기능, 오픈소스, 많은 무료소프트웨어 등등

2가지 이상을 쓰면 5점, 그렇지 않거나 틀린 내용이 있으면 2점

2) (5점) 좀비 프로세스에 대해 설명하시오.

종료된 프로세스로 아직 정리(reap)되지 않은 상태인 프로세스

reaping 청소, 정리 등등이 들어가는 설명이면 5점, 이 단어가 빠지면 2점

3) (5점) 가상 메모리를 사용하는 이유를 2가지 이상 쓰시오.

디스크의 DRAM 캐쉬로 사용, 메모리 관리를 간단하게 함, 보호기능 제공

일부만 맞으면 2점

4) (5점) 동적메모리 할당 라이브러리를 구현할 때 고려해야 하는 두 가지 성능지표를 쓰시오.

Utilization(이용율), Throughput(처리 속도)

일부 맞으면 2점

2. [프로세스 I] (20점) 아래의 프로그램을 실행하면 프로그램이 종료하지 못하고 while 문을 계속 실행하게 된다. 이 때 아래 문제에 답하시오.

```
int ccount = 0;
void child_handler(int sig)
{
      int child_status;
      pid_t pid = wait(&child_status);
      ccount--;
      printf("Received signal %d from process %d\n",
                sig, pid);
    sleep(2);
}

void fork14()
{
      pid_t pid[N];
      int i, child_status;
      ccount = N;
      signal(SIGCHLD, child_handler);
      for (i = 0; i < N; i++)
```

```
        if ((pid[i] = fork()) == 0) {
            sleep(1);
            /* Child: Exit */
            exit(0);
        }
    while (ccount > 0)
        pause();/* Suspend until signal occurs */
}
```

1) (10점) 위 프로그램의 문제점이 무엇이며, 그 이유를 설명하시오.

while 계속 돌고 있으며 종료되지 않는다. 그 이유는 동일한 시그널이 연속해서 발생하고있는데, signal 핸들러에서는 한 개씩 만 처리하고 시그널 비트로만 처리하고 있기 때문이다.

기준 : 시그널이 큐방식으로 처리되지 않는다, 동일한 시그널 처리에 문제가 있다 등이 꼭 들어가면 10점, 일부만 맞으면 5점

2) 1)번의 문제를 해결하는 방법을 쓰시오.

```
void child_handler2(int sig)
{
    int child_status;
    pid_t pid;
    while ((pid = waitpid(-1, &child_status, WNOHANG)) > 0) {
        ccount--;
        printf("Received signal %d from process %d\n", sig, pid);
    }
}
```

핸들러를 위와 같이 수정해서 여러개의 시그널을 동시에 처리 가능하도록 해야 한다.

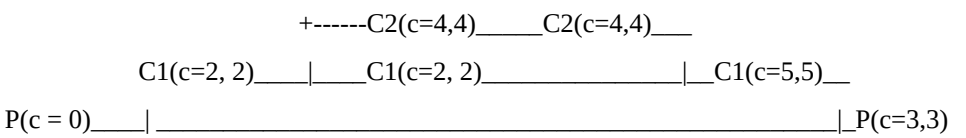
기준 : 프로그램은 없어도 설명이 정확하면 10점. 단순히 프로그램에서 핸들러를 빠르게 처리하도록 한다고 하면 3점, 일부만 맞으면 5점

3. [프로세스 II] (10점) 다음프로그램을 보고 문제에 답하시오.

```
1 int main() {
2    int counter = 0;
3    int pid;
4
5    while (counter < 4 && !(pid = fork())) {
6       counter += 2;
7       printf("%d", counter);
8    }
9
10   if (counter > 0) {
11      printf("%d", counter);
12   }
13
```

```
14   if (pid) {
15      waitpid(pid, NULL, 0);
16      counter += 3;
17      printf("%d", counter);
18   }
29 }
```

1) (5점) 위 프로그램을 실행 시켰을 때, 가능한 출력을 모두 쓰시오.



P = {3}

C1={2,2,5}

C2={4,4}

가능한 출력은

2, {4,4,2}, 5, 3 이다. 따라서 2-442-5-3, 2-424-5-3, 2-244-5-3 이렇게 3가지 출력이 가능하다.

기준: 답을 맞으면 5점, 설명은 안 써도 됨.

2) (5점) 10번째 줄의 > 기호를 >= 로 바꾸면, 가능한 출력이 모두 몇 개가 되는지 쓰시오.

P 가 독립적으로 0을 출력하게 되므로, 기존의 출력과

{0} { 2, {4,4,2}, 5}, 3 이렇게 실행하게 된다. 이 때, 가능한 출력은 1)번의 출력에서 마지막 3을 뺀, 수에서 각각 0을 사이에 넣을 수 있는 경우의 수가 된다. 따라서 예를 들어 244253 에 0을 넣는 방법은 0233253, 20233253, 2303253, 2330253, 2332053, 2332503 이렇게 6가지가 되므로, 총 18가지가 된다.

4. [System I/O] (10 점) 다음 프로그램을 실행했을 때 화면에 출력되는 내용을 쓰시오. “buffer.txt”에는 source 라는 문자열이 저장되어 있다.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
int main() {
       char c;
       int file1 = open("buffer.txt", O_RDONLY);
       int file2;
       read(file1, &c, 1);
       file2 = dup(file1);
       read(file1, &c, 1);
       read(file2, &c, 1);
       printf("1 = %c\n", c);
```

```
        int pid = fork();
        if (pid == 0) {
                close(file1);
                file1 = open("buffer.txt", O_RDONLY);
                read(file1, &c, 1);
                printf("2 = %c\n", c);
                read(file2, &c, 1);
                printf("3 = %c\n", c);
                exit(0);
        } else {
                waitpid(pid, NULL, 0);
                printf("4 = %c\n", c);
                close(file2);
                dup2(file1, file2);
                read(file1, &c, 1);
                printf("5 = %c\n", c);
                read(file2, &c, 1);
                printf("6 = %c\n", c);
        }
        return 0;
}
```

그림을 그리기에도 다소 복잡함. 꼼꼼히 생각하면 아래와 같은 답이 나옴.

1=u, 2=s, 3=r, 4=u, 5=c, 6=e

기준 : 1~2번은 각 1점, 나머지는 각 2점

5. [Longjump, Signal] (10pts) 아래의 프로그램은 일반적인 fgets 함수를 timeout 기능을 추가한 것으로, tfgets(buf, sizeof(buf), stdin) 으로 실행시킬 때, 사용자가 리턴키를 5초동안 입력하지 않게 되면 SIGUSR2 시그널을 발생시켜서 핸들러에서 siglongjump를 부르게 되며, 결국 NULL을 리턴하도록 하려고 한다. 빈 곳 (1), (2) 을 채워서 프로그램을 완성하시오.

```
static sigjmp_buf env;
void handler(int sig) {
        wait(NULL);
        _________________  ; (1) 번
}
char *tfgets(char *s, int size, FILE *stream)
{
        pid_t pid;
        signal(SIGUSR2, handler);
        if (sigsetjmp(env, 1) != 0)
                return NULL;
        if ((pid = fork()) == 0) {
                sleep(5);
                _________________ ; (2) 번
                exit(0);
        }
        fgets(s, size, stream);
        kill(pid, SIGKILL);
```

```
        wait(NULL);
        return s;
}
```

```
(1) : siglongjmp(env,1);
(2) : kill(getppid(), SIGUSR2);
```

기준 : 각 5점, 철자가 틀리거나 문법이 틀려도 채점 가능. (1) 번은 longjump(env)만 써도 맞게 채점 (2)번은 정확히 맞아야 함

6.[동적메모리 할당 I](10점) 동적메모리 할당 라이브러리를 구현하기 위해 다음과 같은 alignment 조건과 블록 설계를 하는 경우에 아래표의 최소 블럭 크기를 결정하시오. 간접리스트 방식으로 구현하며, payload의 크기는 0보다 커야 하고, header와 footer는 각각 4바이트 워드 크기로 저장된다.

Alignment	할당된 블록	Free 블록	최소 블록 크기(바이트)
Single-word	Header, footer	Header, footer	12바이트
Double-word	Header	Header	8 바이트

기준 : 각 5점씩. 각각 정확히 맞아야 함

7. [동적메모리 할당 II](10점) 다음과 같은 순서로 malloc 요청을 하는 경우 블록의 크기와 header에 저장되는 값을 쓰시오. 헤더는 4바이트를 사용하며, 비트 0 는 할당/free를 표시하는데 사용한다. 할당은 8 바이트, 즉 double word alignment 를 유지해야 한다.

요청	블록크기(바이트)	블록 헤더 값(hex)
malloc(5)	16	0x11
malloc(15)	24	0x19

기준 : 각 2.5점 후에 반올림, 각각 정확히 맞아야 함

8. [동적메모리 할당 III] (20점) 다음 질문에 답하시오.

1) (5점) 간접리스트 방식에서 Free 메모리 블럭을 찾아서 할당하려고 할 때 First Fit, Next Fit, Best Fit 의 세 가지 방식을 고려하고 있다. 이 중에서 단편화(Fragmentation)을 최소화 할 수 있는 방법은 어느 것인가 ?

Best Fit, 일부 점수 없음

2) (5점) 간접리스트 방식에서 Free 메모리를 할당할 때, 내부 단편화를 줄이기 위해 해주어야 하는 작업은 무엇인가?

할당 후 남는 메모리는 블럭 쪼개기를 한다.(Splitting)

3) (5점) 간접리스트 방식에서 할당되었던 메모리를 리턴시킬때(free()) 외부 단편화를 줄이기

위해서 해주어야 하는 작업은 무엇인가 ?
이전, 이후 블럭이 free 하면 블럭을 합쳐준다. Coalescing 한다
4) (5점) 양방향 포인터를 사용하는 직접 리스트 방식으로 malloc( )을 구현할 때 free블럭을 검색하는 데 소요되는 시간을 간접 리스트 방식에서와 비교해서 어느 쪽이 더 빠른지 설명하고, 그 이유를 설명하시오.
직접리스트에서 최악 검색시간은 총 free 블럭의 갯수가 된다. 간접리스트에서는 모든 블럭의 갯수만 큼 최악 검색시간이 소요하므로, 직접리스트가 검색 시간이 더 짧아진다.