1.기본문제

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
    pid_t pid;
    pid = fork();
        if(pid<0){
                fprintf(stderr,"Fork Failed");
                return 1;
        }

        else if(pid==0){                //자식 프로세스 들어감
                printf("child PID : %d\n",pid);
                execlp("ps","ps -al",NULL);
        }

        else {
                printf("parent PID : %d\n",pid);
                wait(NULL);
                printf("Child Complete");
        }
        return 0;
}
~
~
```

```
hunterspin@ubuntu:~/Desktop$ vi fork.c
hunterspin@ubuntu:~/Desktop$ gcc -o fork fork.c
hunterspin@ubuntu:~/Desktop$ ./fork
parent PID : 2179
child PID : 0
    PID TTY          TIME CMD
   1939 pts/0    00:00:00 bash
   2178 pts/0    00:00:00 fork
   2179 pts/0    00:00:00 ps
Child Completehunterspin@ubuntu:~/Desktop$
```

2.도전과제

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    pid_t pid;
    int pid_array[4];
    for (int i = 0; i < 4; i++)
    {
        srand((unsigned int)time(NULL)); // srand 함수를 이용하여 랜덤값 시
드 수행할때마다 변경
        pid_array[i] = fork();          //자식 프로세스 생성
        if (pid_array[i] < 0)
        { // 에러 처리
            return -1;
        }
        else if (pid_array[i] == 0)
        {                           // 자식 프로세스 부분
            sleep(rand() % 16 + 5); // 5-20초 랜덤하게 sleep
            if (i == 0)
            {                               //첫번째 자식프로세스
                execlp("ps", "ps", NULL); // ps명령어 실행
                pid_array[i] = getpid();  //배열에 pid값 저장
                exit(0);                  //종료
            }
            else if (i == 1)
            {                               //두번째 자식프로세스
                execlp("ls", "ls", NULL); // ls명령어 실행
                pid_array[i] = getpid();  //배열에 pid값 저장
                exit(1);                  //종료
            }
            else if (i == 2)
            {                               //세번째 자식프로세스
                execlp("df", "df", NULL); // ls명령어 실행
                pid_array[i] = getpid();  //배열에 pid값 저장
```

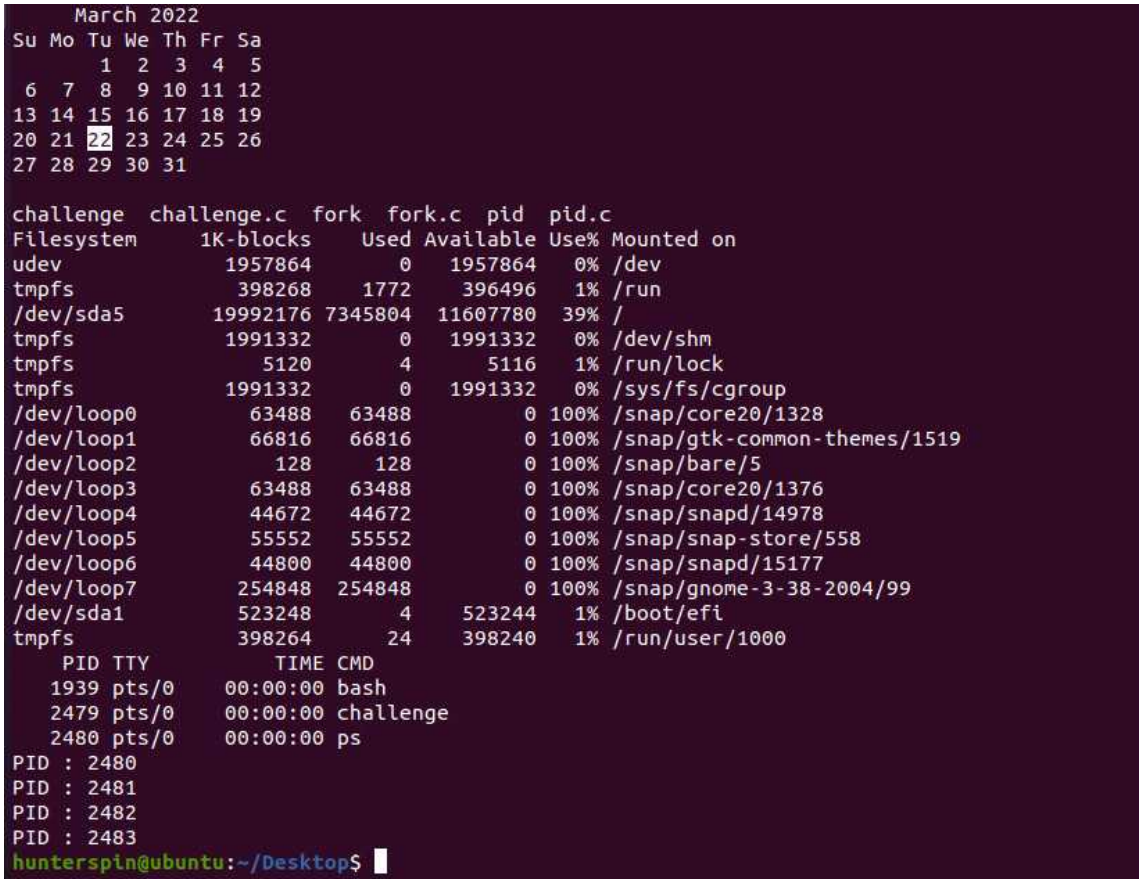
```
                exit(2);                    //종료
            }
            else if (i == 3)
            {                                //네번째 자식프로세스
                execlp("cal", "cal", NULL); // ls명령어 실행
                pid_array[i] = getpid();    //배열에 pid값 저장
                exit(3);                    //종료
            }                                // exit하지 않았을경우 2^(fork실
행횟수)만큼 자식프로세스 생성
        }
        else
        { //부모 프로세스 부분
        }
    }
    while (wait(NULL) > 0)
        ; //부모가 자식이 종료될때까지 기다리는 부분
    for (int i = 0; i < 4; i++)
    { //배열에 저장된값 모두 출력
        printf("PID : %d\n", pid_array[i]);
    }
    return 0;
}
```

5-20초 랜덤하게 멈춘 뒤 PID값 출력

```
hunterspin@ubuntu:~/Desktop$ vi challenge.c
hunterspin@ubuntu:~/Desktop$ gcc -o challenge challenge.c
hunterspin@ubuntu:~/Desktop$ ./challenge
challenge  challenge.c  fork  fork.c  pid  pid.c
Filesystem     1K-blocks    Used Available Use% Mounted on
udev             1957864       0   1957864   0% /dev
tmpfs             398268    1776    396492   1% /run
/dev/sda5       19992176 7345800  11607784  39% /
tmpfs            1991332       0   1991332   0% /dev/shm
tmpfs               5120       4      5116   1% /run/lock
tmpfs            1991332       0   1991332   0% /sys/fs/cgroup
/dev/loop0         63488   63488         0 100% /snap/core20/1328
/dev/loop1         66816   66816         0 100% /snap/gtk-common-themes/1519
/dev/loop2           128     128         0 100% /snap/bare/5
/dev/loop3         63488   63488         0 100% /snap/core20/1376
/dev/loop4         44672   44672         0 100% /snap/snapd/14978
/dev/loop5         55552   55552         0 100% /snap/snap-store/558
/dev/loop6         44800   44800         0 100% /snap/snapd/15177
/dev/loop7        254848  254848         0 100% /snap/gnome-3-38-2004/99
/dev/sda1         523248       4    523244   1% /boot/efi
tmpfs             398264      24    398240   1% /run/user/1000
     March 2022
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

    PID TTY          TIME CMD
   1939 pts/0    00:00:00 bash
   2465 pts/0    00:00:00 challenge
   2466 pts/0    00:00:00 ps
PID : 2466
PID : 2467
PID : 2468
PID : 2469
hunterspin@ubuntu:~/Desktop$
```

위의 실행결과와 아래 실행결과가 출력되는 순서가 다르다.


March 2022
Su Mo Tu We Th Fr Sa
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
challenge challenge.c fork fork.c pid pid.c
Filesystem 1K-blocks Used Available Use% Mounted on
udev 1957864 0 1957864 0% /dev
tmpfs 398268 1772 396496 1% /run
/dev/sda5 19992176 7345804 11607780 39% /
tmpfs 1991332 0 1991332 0% /dev/shm
tmpfs 5120 4 5116 1% /run/lock
tmpfs 1991332 0 1991332 0% /sys/fs/cgroup
/dev/loop0 63488 63488 0 100% /snap/core20/1328
/dev/loop1 66816 66816 0 100% /snap/gtk-common-themes/1519
/dev/loop2 128 128 0 100% /snap/bare/5
/dev/loop3 63488 63488 0 100% /snap/core20/1376
/dev/loop4 44672 44672 0 100% /snap/snapd/14978
/dev/loop5 55552 55552 0 100% /snap/snap-store/558
/dev/loop6 44800 44800 0 100% /snap/snapd/15177
/dev/loop7 254848 254848 0 100% /snap/gnome-3-38-2004/99
/dev/sda1 523248 4 523244 1% /boot/efi
tmpfs 398264 24 398240 1% /run/user/1000
PID TTY TIME CMD
1939 pts/0 00:00:00 bash
2479 pts/0 00:00:00 challenge
2480 pts/0 00:00:00 ps
PID : 2480
PID : 2481
PID : 2482
PID : 2483
hunterspin@ubuntu:~/Desktop$

■보고서

1.목적

- fork함수를 이해하고 사용할줄 알며 자식프로세스와 부모프로세스의 관계에 대해서 이해하고 프로그래밍 할 수 있는 방법을 알기위해서

2.과정

-fork에 관해 구글링과 영상을 통해 학습한 뒤, 그에 따라 자식프로세스의 동작 원리에 대해 알게 되었다. 그리고 exec함수라고 하는 것에 대해서 알게 되었고 프로세스의 동시 동작 원리에 대해서도 알게 되었다.
랜덤으로 sleep걸라고 문제에 적혀있어 rand함수로 시도해보았지만, 계속해서 순서가 똑같이 나와 무엇이 문제인지 생각해보았는데, 결국 srand라는 함수를 써서 시드 값을 바꾸지 않으면 일정한 시드값으로 계속 돌아간다는 것을 알게되었다.

■결과

-프로그래밍 결과

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    pid_t pid;
    int pid_array[4];
    for (int i = 0; i < 4; i++)
    {
```

```
        srand((unsigned int)time(NULL)); // srand 함수를 이용하여 랜덤값 시
드 수행할때마다 변경
        pid_array[i] = fork();           //자식 프로세스 생성
        if (pid_array[i] < 0)
        { // 에러 처리
            return -1;
        }
        else if (pid_array[i] == 0)
        {                           // 자식 프로세스 부분
            sleep(rand() % 16 + 5); // 5-20초 랜덤하게 sleep
            if (i == 0)
            {                               //첫번째 자식프로세스
                execlp("ps", "ps", NULL); // ps명령어 실행
                pid_array[i] = getpid();  //배열에 pid값 저장
                exit(0);                  //종료
            }
            else if (i == 1)
            {                               //두번째 자식프로세스
                execlp("ls", "ls", NULL); // ls명령어 실행
                pid_array[i] = getpid();  //배열에 pid값 저장
                exit(1);                  //종료
            }
            else if (i == 2)
            {                               //세번째 자식프로세스
                execlp("df", "df", NULL); // ls명령어 실행
                pid_array[i] = getpid();  //배열에 pid값 저장
                exit(2);                  //종료
            }
            else if (i == 3)
            {                                 //네번째 자식프로세스
                execlp("cal", "cal", NULL); // ls명령어 실행
                pid_array[i] = getpid();    //배열에 pid값 저장
                exit(3);                    //종료
            }                                 // exit하지 않았을경우 2^(fork실
행횟수)만큼 자식프로세스 생성
        }
        else
        { //부모 프로세스 부분
        }
```

```
    }
    while (wait(NULL) > 0)
        ; //부모가 자식이 종료될때까지 기다리는 부분
    for (int i = 0; i < 4; i++)
    { //배열에 저장된값 모두 출력
        printf("PID : %d\n", pid_array[i]);
    }
    return 0;
}
```

목표했던 것은 프로세스 하나가 종료될때마다 프로세스ID를 출력해 주는 것이었는데 이렇게 하려면 부모프로세스 처리부분에 출력을 넣어주면 되기는 하는데,그럴 경우 모든 프로세스가 동시에 시작하지 않는다는 단점이 생겨 한번에 출력하는 것으로 바꾸었다.

아래 코드는 프로세스마다 하나씩 프로세스 번호를 출력해준다.

-단점이 있는 코드-

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    pid_t pid;
    int pid_array[4];
    int status;
    int pid_child;
    for (int i = 0; i < 4; i++)
    {
        srand((unsigned int)time(NULL));
        pid_array[i] = fork();
        if (pid_array[i] < 0)
        { // 에러 처리
            return -1;
        }
```

```
        else if (pid_array[i] == 0)
        { // 자식 프로세스 부분
            sleep(rand() % 16 + 5);
            if (i == 0)
            {
                pid_array[0] = getpid();
                execlp("ps", "ps", NULL);
                exit(0);
            }
            else if (i == 1)
            {
                pid_array[1] = getpid();
                execlp("ls", "ls", NULL);
                exit(1);
            }
            else if (i == 2)
            {
                pid_array[2] = getpid();
                execlp("df", "df", NULL);
                exit(2);
            }
            else if (i == 3)
            {
                pid_array[3] = getpid();
                execlp("cal", "cal", NULL);
                exit(3);
            }
        }
        else
        {
            printf("PID : %d\n", pid_array[i]);
        }
        while (wait(NULL) > 0);
    }
    return 0;
```