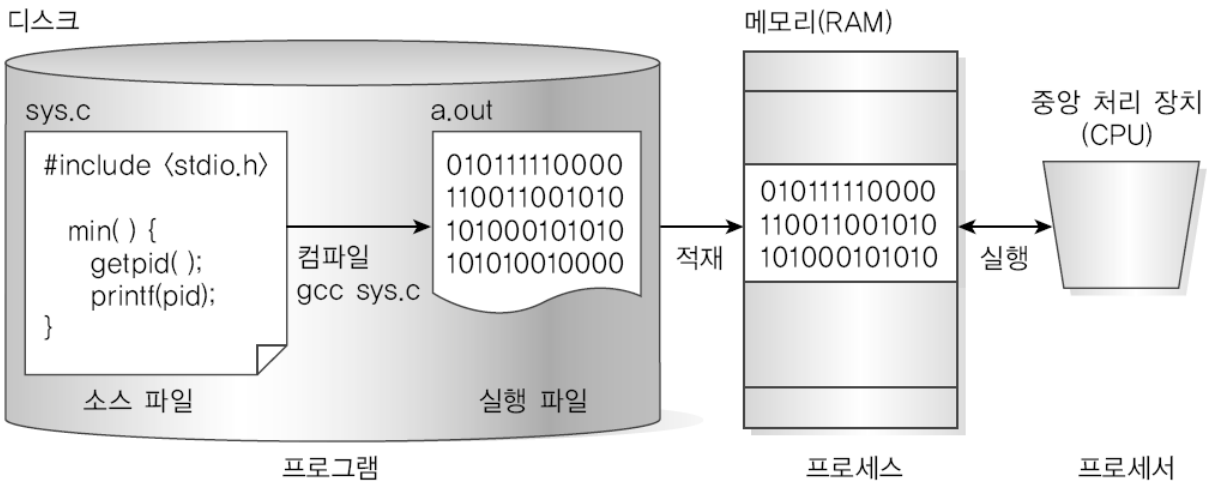
Process 1
(fork)

Jo, Heeseung

프로세스의 정의

프로세스 (process)

- 실행중인 프로그램
  - 프로세서(processor) : 중앙처리장치(예: 펜티엄, 쿼드코어 등)
  - 프로그램(program) : 사용자가 컴파일에 작업을 시키기 위한 명령어의 집합 (스토리지에 존재)
- 메모리에 존재



[그림 5-1] 프로그램, 프로세스, 프로세서의 관계

프로세스 목록 보기

현재 실행중인 프로세스 목록을 보려면 ps 명령 사용

```
# ps
  PID TTY          TIME CMD
  678 pts/3        0:00 ksh
 1766 pts/3        0:00 ps
```

```
# ps -ef ¦ more
  UID   PID  PPID   C    STIME    TTY  TIME  CMD
root     0     0   0    1월 30일  ?    175:28 sched
root     1     0   0    1월 30일 ?     0:02  /sbin/init
root     2     0   0    1월 30일 ?     0:00  pageout
.......
```

현재 실행중인 프로세스를 주기적으로 확인

- 솔라리스 기본 명령 : prstat, sdtprocess
- 공개소프트웨어 : top, htop

프로세스 식별

PID 검색: getpid(2)

```
#include <unistd.h>
pid_t getpid(void);
```

- 이 함수를 호출한 프로세스의 PID를 리턴

PPID 검색 : getppid(2)

```
#include <unistd.h>
pid_t getppid(void);
```

- 부모 프로세스의 PID를 리턴

```
# ps -ef | more
 UID   PID  PPID   C    STIME    TTY  TIME  CMD
root     0     0   0    1월 30일  ?    175:28 sched
root     1     0   0    1월 30일  ?    0:02  /sbin/init
root     2     0   0    1월 30일  ?    0:00  pageout
......
```

부모 프로세스ID

getpid, getppid 함수 사용하기

```
01  #include <unistd.h>
02  #include <stdio.h>
03
04  int main(void) {
05      printf("PID : %d\n", (int)getpid());
06      printf("PPID : %d\n", (int)getppid());
07
08      return 0;
09  }
```

```
# ex5_1.out
PID : 2205
PPID : 678
```

```
# ps
PID TTY          TIME CMD
678 pts/3        0:00 ksh
2206 pts/3       0:00 ps
```

프로세스 실행 시간 측정

프로세스 실행 시간의 구성

프로세스 실행시간 = 시스템 실행시간 + 사용자 실행시간

- 시스템 실행시간
  - 커널 코드를 수행한 시간(시스템 호출로 소비한 시간)
- 사용자 실행시간
  - 사용자 모드에서 프로세스를 실행한 시간

프로세스 실행 시간 측정

프로세스 실행 시간 측정

```
#include <sys/times.h>
#include <limits.h>

clock_t times(struct tms *buffer);
```

- 사용자와 시스템 실행시간으로 나누어 tms 구조체에 저장
- 시간 단위는 클록틱(sysconf 함수에서 _SC_CLK_TCK로 검색한 값)
- tms 구조체

```
struct tms {
    clock_t tms_utime;
    clock_t tms_stime;
    clock_t tms_cutime;
    clock_t tms_cstime;
};
```

```
utime : 사용자 모드실행시간
stime : 시스템 모드실행시간
cutime : 자식프로세스의 사용자 모드 실행시간
cstime : 자식프로스세의 시스템 모드 실행시간
```

time 명령 사용

time(1)

- 특정 프로그램을 수행하고, 시스템 리소스 사용율을 보여줌
- -p 옵션을 추가하면 초 단위 형식으로만 출력

- time ls -aFl
- time ./gugudan
- time -p sleep 2

프로세스 실행 시간 측정 2

프로세스 실행 시간 측정 2

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

int gettimeofday(struct timeval *tv, struct timezone *tz);
```

- 현재 시간을 microsecond 단위로 측정
- 보다 정확한 시간을 측정

```
struct timeval {
  long    tv_sec;         // seconds
  long    tv_usec;        // and microseconds
};
```

gettimeofday 함수 사용하기

```c
#include <stdio.h>
#include <sys/time.h>

int main(void) {
    int i;
    struct timeval t1, t2;

    gettimeofday(&t1, NULL);

    for (i = 0; i < 9999999; i++)
       rand();

    gettimeofday(&t2, NULL);

    printf("%f\n", t1.tv_sec + t1.tv_usec*0.000001);
    printf("%f\n", t2.tv_sec + t2.tv_usec*0.000001);
    printf("%f\n", (t2.tv_sec + t2.tv_usec*0.000001)-(t1.tv_sec + t1.tv_usec*0.000001) );

    return 0;
}
```

```
# ./a.out
1366252562.446275
1366252562.454634
0.008359

# time ./a.out
1366253017.697086
1366253017.705058
0.007972

real    0m0.010s
user    0m0.004s
sys     0m0.004s
```

trace Macro

```c
#include <stdio.h>

#define trs(x...) { printf("[%s:%d] %s = ", __func__, __LINE__, #x); printf("%s\n", x); }
#define tri(x...) { printf("[%s:%d] %s = ", __func__, __LINE__, #x); printf("%d\n", x); }

int main(void) {
        int i=1024;
        char buf[255]="abcdefg";

        trs(buf);
        tri(i);
        tri(1);

        return 0;
}
```

```
root@iter1:/tmp> ./a.out
[main:10] buf = abcdefg
[main:11] i = 1024
[main:12] 1 = 1
```

Exercise

Ex.

- 구구단 출력 프로그램을 수정하여 100단까지 출력하는데 걸리는 시간을 측정하여 보자
- gettimeofday() 함수 이용

```
# ./gugudan
2 x 1 = 2
...
...
100 x 9 = 900
Execution time : 0.008359s
```

프로세스 생성[1]

프로그램 실행 : system(3)

```
#include <stdlib.h>
int system(const char *string);
```

- 새로운 프로그램을 실행하는 가장 간단한 방법
- 실행할 프로그램명을 인자로 지정

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      int a;
06      a = system("ps -ef | grep bash > result.txt");
07      printf("Return Value : %d\n", a);
08
09      return 0;
10  }
```

```
# ex6_1.out

Return Value : 0
```

프로세스 생성[2]

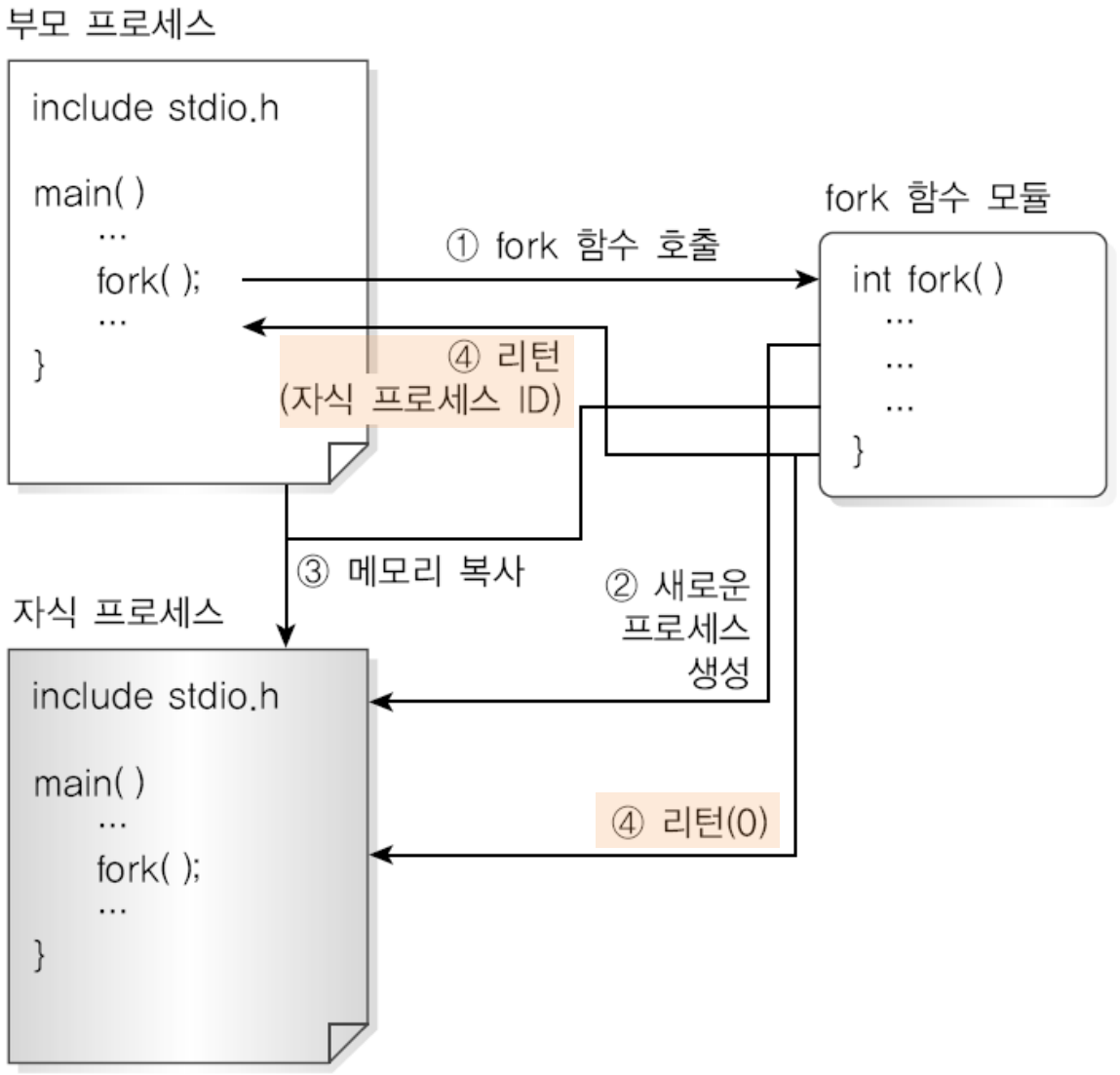
프로세스 생성: fork(2)

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

- 새로운 프로세스를 생성 : 자식 프로세스 (return 0)
- fork 함수를 호출한 프로세스 : 부모 프로세스 (return ChildPID)
- 자식 프로세스는 부모 프로세스의 메모리를 복사
  - RUID, EUID, RGID, EGID, 환경변수
  - 열린 파일기술자, 시그널 처리, setuid, setgid
  - 현재 작업 디렉토리, umask, 사용가능자원 제한
- 부모 프로세스와 자식 프로세스는 열린 파일을 공유
  - Read/Write에 주의

프로세스 생성[2]

프로세스 생성



[그림 6-1] fork 함수를 이용한 새로운 프로세스 생성

fork 함수 사용하기

```
...
06  int main(void) {
07      pid_t pid;
08
09      switch (pid = fork()) {
10          case -1 :  /* fork failed */
11              perror("fork");
12              exit(1);
13              break;
14          case 0 :   /* child process */
15              printf("Child Process - My PID:%d, My Parent's PID:%d\n",
16                  (int)getpid(), (int)getppid());
17              break;
18          default : /* parent process */
19              printf("Parent process - My PID:%d, My Parent's PID:%d,
                    My Child's PID:%d\n", (int)getpid(), (int)getppid(),
                    (int)pid);
21              break;
22      }
23
24      printf("End of fork\n");
26      return 0;
27  }
```

```
# ex6_2.out
Child Process - My PID:796, My Parent's PID:795
End of fork
Parent process - My PID:795, My Parent's PID:695,
My Child's PID:796
End of fork
```

fork함수의 리턴값 0은
자식 프로세스가 실행

Exercise

Ex.

- Parent process는 2-5단, child process는 6-9단을 출력하는 프로그램을 작성

```
./a.out

Parent process - My PID:15695, My Parent's PID:15602, My
Child's PID:15696
2 x 1 = 2
2 x 2 = 4
Child Process - My PID:15696, My Parent's PID:15695
6 x 1 = 6
2 x 3 = 6
2 x 4 = 8
6 x 2 = 12                (순서가 섞여서 출력됨)
6 x 3 = 18
6 x 4 = 24
...
```

Exercise

Ex.

- 100,000까지의 모든 prime number를 출력하는 psingle 프로그램을 작성
  - time 명령을 이용하여 몇 초가 걸리는지 확인할 것

- 위의 프로그램과 동일하지만 4개의 프로세스를 이용하는 pmulti 프로그램을 작성
  - 4개의 프로세스는 구간을 나누어서 병렬처리
  - time 명령을 이용하여 몇 초가 걸리는지 확인할 것