KWANGWOON UNIVERSITY

2024년 2학기 **운영체제실습**

Assignment 3

System Software Laboratory
School of Computer and Information Engineering
Kwangwoon Univ.

Contents

- **Assignment3-1**
  - Task information modules
- **Assignment3-2**
  - Fork / Thead analysis
- **Assignment3-3**
  - CPU scheduler simulator

Assignment 3-1

- **PID를 바탕으로 아래와 같은 프로세스의 정보를 출력하는 Module 작성**
  - (1) 프로세스 이름
  - (2) 현재 프로세스의 상태
    - Running or ready, Wait with ignoring all signals, Wait, Stopped, Zombie process, Dead, etc.
  - (3) 프로세스 그룹 정보
    - PGID, 프로세스 이름
  - (4) 해당 프로세스를 실행하기 위해 수행된 context switch 횟수
  - (5) fork()를 호출한 횟수
  - (6) 부모(parent) 프로세스 정보
    - PID, 프로세스 이름
  - (7) 형제자매(sibling) 프로세스 정보
    - PID, 프로세스 이름, 총 형제자매 프로세스 수
  - (8) 자식(child) 프로세스 정보
    - PID, 프로세스 이름, 총 자식 프로세스 수

Assignment 3-1

- **PID를 바탕으로 아래와 같은 프로세스의 정보를 출력하는 Module 작성**
  - Example

```
[49474.036512] ##### TASK INFORMATION of ''[1] systemd'' #####          (1)
[49474.036513] - task state : Wait with ignoring all signals            (2)
[49474.036528] - Process Group Leader : [1] systemd                     (3)
[49474.036529] - Number of context switches : 7607                      (4)
[49474.036529] - Number of calling fork() : 378                         (5)
[49474.036530] - it's parent process : [0] swapper/0                    (6)
[49474.036530] - it's sibling process(es) :                             (7)
[49474.036531]    > [385] systemd-journal
[49474.036531]    > [439] vmtoolsd
[49474.036532]    > [437] vmware-vmblock-
[49474.036532]    > [852] acpid
[49474.036533]    > [855] accounts-daemon
                          .
                          .
                          .
[49474.036552]    > [22802] systemd-udevd
[49474.036552]    > [23193] cupsd
[49474.036552]    > [23194] cups-browsed
[49474.036553]    > [23798] whoopsie
[49474.036553]    > [24093] polkitd
[49474.036553]    > This process has 28 child process(es)               (8)
[49474.036554] ##### END OF INFORMATION #####
```

Assignment 3-1

- **Requirements**
  - 2차 과제에서 작성한 ftrace 시스템 콜(336번)을 다음 함수로 **wrapping 하여 사용**
    - **asmlinkage pid_t process_tracer(pid_t trace_task);**
      - Return value
        - 정상적으로 정보를 출력한 경우; 입력 받은 PID 값
        - 출력을 완료하지 못한 경우; -1
      - Input value
        - pid_t trace_task : 정보를 출력할 프로세스의 ID 값
    - 예시

시스템 콜 wrapping 후

```
1 #include <linux/unistd.h>
2
3 int main(void)
4 {
5     syscall(__NR_ftrace, 1);
6     return 0;
7 }
```

```
[49474.036512] ##### TASK INFORMATION of ''[1] systemd'' #####
[49474.036513] - task state : Wait with ignoring all signals
[49474.036528] - Process Group Leader : [1] systemd
[49474.036529] - Number of context switches : 7607
[49474.036529] - Number of calling fork() : 378
[49474.036530] - it's parent process : [0] swapper/0
[49474.036530] - it's sibling process(es) :
[49474.036531]     > [385] systemd-journal
[49474.036531]     > [439] vmtoolsd
[49474.036532]     > [437] vmware-vmblock-
[49474.036532]     > [852] acpid
[49474.036533]     > [855] accounts-daemon
                    ⋮
[49474.036552]     > [22802] systemd-udevd
[49474.036552]     > [23193] cupsd
[49474.036552]     > [23194] cups-browsed
[49474.036553]     > [23798] whoopsie
[49474.036553]     > [24093] polkitd
[49474.036553]     > This process has 28 child process(es)
[49474.036554] ##### END OF INFORMATION #####
```

Assignment 3-1

- **Requirements**
  - 메시지 출력 형식
    - Kernel ring buffer에 본 자료에 포함된 예시와 같은 형태로 출력
    - 프로세스 이름은 축약하지 않은 형태로 출력
  - 현재 프로세스의 상태
    - 다음과 같은 7가지의 상태로 구분
      - Running or ready
      - Wait with ignoring all signals
      - Wait
      - Stopped
      - Zombie process
      - Dead
      - etc.

Assignment 3-1

- **Requirements**
  - Context switch 횟수

  - fork() 호출한 횟수
    - 참고: 구현 방법
    - - task_struct 구조체에 fork() 호출 횟수를 저장하는 변수를 추가한다.
    - - fork()로 자식프로세스를 생성할 때 해당 변수를 초기화 시킨다.
    - - fork()가 호출될 때 마다 해당 변수의 값을 1씩 증가시킨다.

Assignment 3-1

- **Requirements**
  - 부모(parent) 프로세스 정보
    - 부모 프로세스의 정보를 출력한다.
  - 형제자매(sibling) 프로세스 정보
    - 자기 자신을 제외한 형제자매 프로세스의 정보를 출력한다.

```c
#include <linux/unistd.h>
#include <unistd.h>

int main(void)
{
    syscall(__NR_ftrace, getpid());
    return 0;
}
```

```
[82701.953972] [OSLab.] ##### TASK INFORMATION of '[8138] app' #####
[82701.953976] [OSLab.]   - task state : Running or ready
[82701.953977] [OSLab.]   - # of context-switch(es) : 2
[82701.953978] [OSLab.]   - its parent process : [7934] make
[82701.953980] [OSLab.]   - its sibling process(es) :
[82701.953981] [OSLab.]     > It has no sibling.
```

- 마지막에 총 형제자매 프로세스 수를 출력한다.

```
[66334.057472] [OSLab.]   - its sibling process(es) :
[66334.057473] [OSLab.]     > [2] kthreadd
[66334.057475] [OSLab.]     > This process has 1 sibling process(es)
```

Assignment 3-1

- **Requirements**
  - “자식 프로세스 정보”
    - 자식 프로세스의 정보를 출력한다.
    - 없을 경우, 다음과 같이 출력한다.

```c
#include <linux/unistd.h>
#include <unistd.h>

int main(void)
{
    syscall(__NR_ftrace, getpid());
    return 0;
}
```

```
[82701.953972] [OSLab.] ##### TASK INFORMATION of '[8138] app' #####
[82701.953976] [OSLab.]  - task state : Running or ready
[82701.953977] [OSLab.]  - # of context-switch(es) : 2
[82701.953978] [OSLab.]  - its parent process : [7934] make
[82701.953980] [OSLab.]  - its sibling process(es) :
[82701.953981] [OSLab.]    > It has no sibling.
[82701.953982] [OSLab.]  - its child process(es) :
[82701.953983] [OSLab.]    > It has no child.
[82701.953984] [OSLab.] ##### END OF INFORMATION #####
```

  - 마지막에 총 자식 프로세스 수를 출력한다.

```
[66334.057621] [OSLab.]    > [1652] indicator-application-service
[66334.057623] [OSLab.]    > This process has 45 child process(es)
[66334.057624] [OSLab.] ##### END OF INFORMATION #####
```

Assignment 3-2

- **다음의 작업을 다중 프로세스/쓰레드로 수행하는 프로그램을 작성**
  - 프로그램 1. **numgen** (numgen.c)
    - Step 1. 특정 파일(“./temp.txt”)를 생성
    - Step 2. **i번째 값을 integer형 양수으로 생성할 프로세스 수의 2배만큼(MAX_PROCESSES)** 기록

```
…
#define MAX_PROCESSES 4
…
   FILE *f_write = fopen(...

   for(i=0; i<MAX_PROCESSES*2; i++)
   {
      fprintf(f_write, "%d", i+1);
      …
   }

   fclose(f_write);
…
```

temp.txt

```
1
2
3
4
5
6
7
8
```

Assignment 3-2

- **다음의 작업을 다중 프로세스/쓰레드로 수행하는 프로그램을 작성**
  - 프로그램 2. fork.c, thread.c
    - Step 1. **MAX_PROCESSES** 만큼 프로세스 또는 스레드를 생성
      - **MAX_PROCESSES = 8, 64**
    - Step 2. 최상단 프로세스/스레드마다 2개의 숫자를 읽음.
    - Step 3. 각 프로세스/스레드는 두 개의 숫자를 더한 후,
      부모 프로세스/스레드에게 값을 전달 (fork → **exit() 사용**)
    - Step 4. 최종적으로 나온 값, 전체 프로그램 수행시간 측정
      - clock_gettime() 사용
      - **결과에 대한 분석 내용 작성**

```
void main(void){
        ⋮
clock_gettime(                    );
if(!fork()){
        ⋮
}
wait(&result);
clock_gettime(                    );
        ⋮
}
```

  - 프로세스를 수행하는 파일(fork.c), 스레드로 수행하는 파일(thread.c) 각각 구현

Assignment 3-2

- **다음의 작업을 다중 프로세스/쓰레드로 수행하는 프로그램을 작성**
  - Step 4. 최종적으로 나온 값, 전체 프로그램 수행시간 측정

```
os2019110613@ubuntu:~/test2$ ./a.out
value of fork : 136
0.002265
```

```
os2019110613@ubuntu:~/test2$ ./a.out
value of thread : 136
0.002423
```

< MAX_PROCESS = 8 >

```
os2019110613@ubuntu:~/test2$ ./a.out
value of fork : 64
0.016572
```

```
os2019110613@ubuntu:~/test2$ ./a.out
value of thread : 8256
0.013368
```

< MAX_PROCESS = 64 >

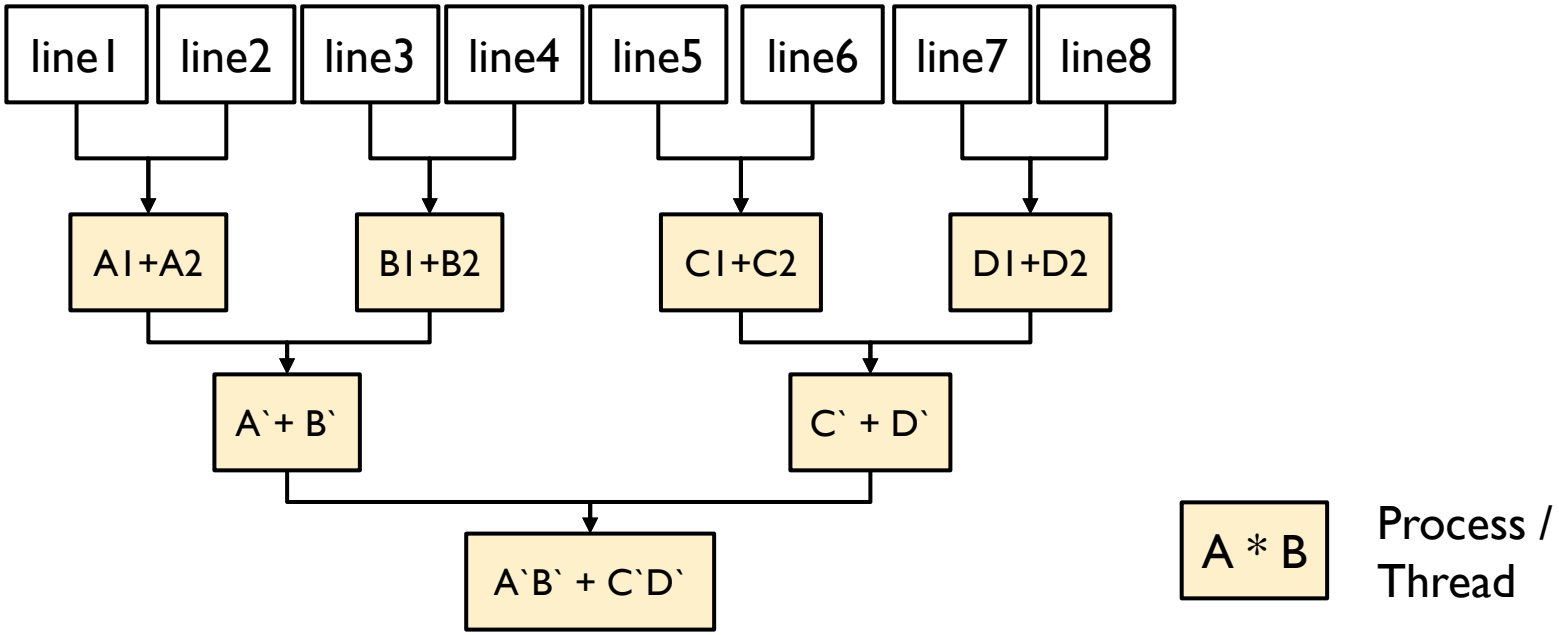
  - **결과에 대한 분석 내용 작성**

Assignment 3-2

<Example>

- **filegen.c**
  - **Ex) MAX_PROCESS가 4이면, 숫자 8개를 기록**

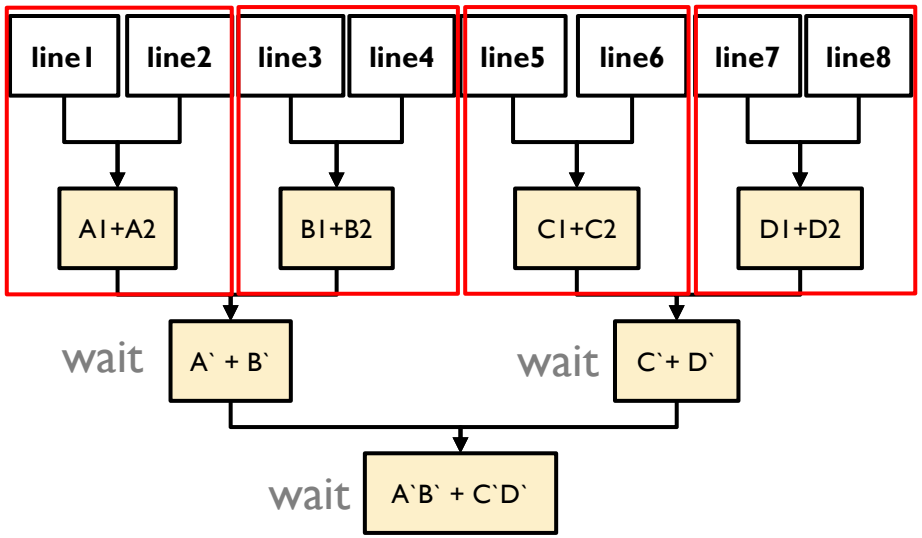
- **fork.c / thread.c**



Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



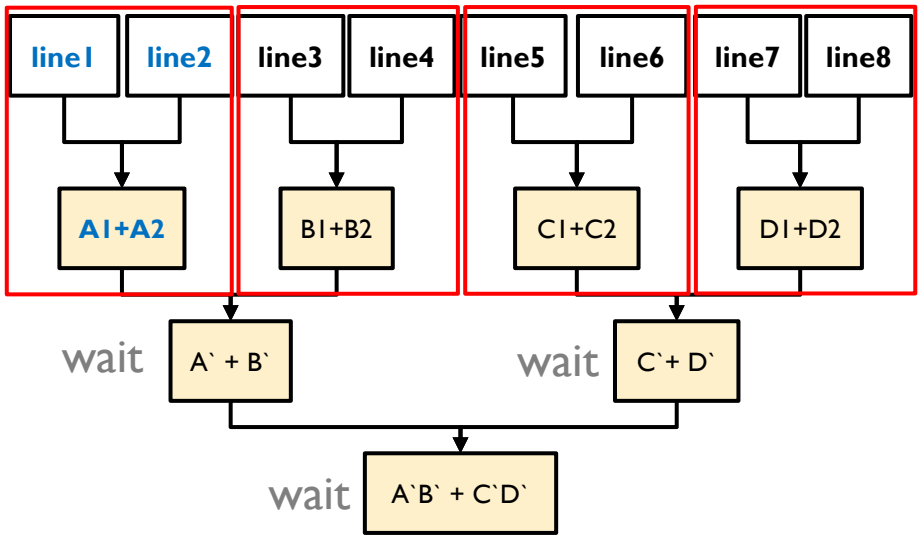
temp.txt

1
2
3
4
5
6
7
8

Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



temp.txt

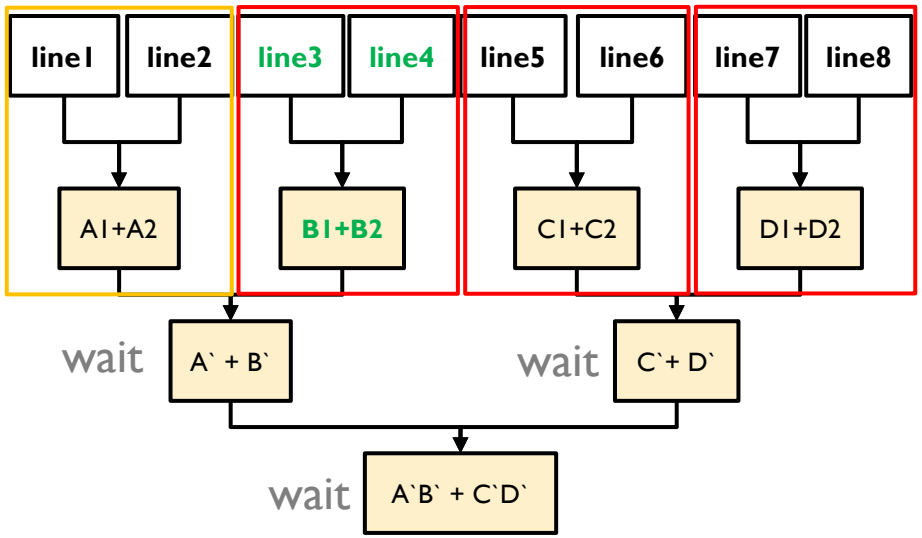
1
2
3
4
5
6
7
8

3

Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



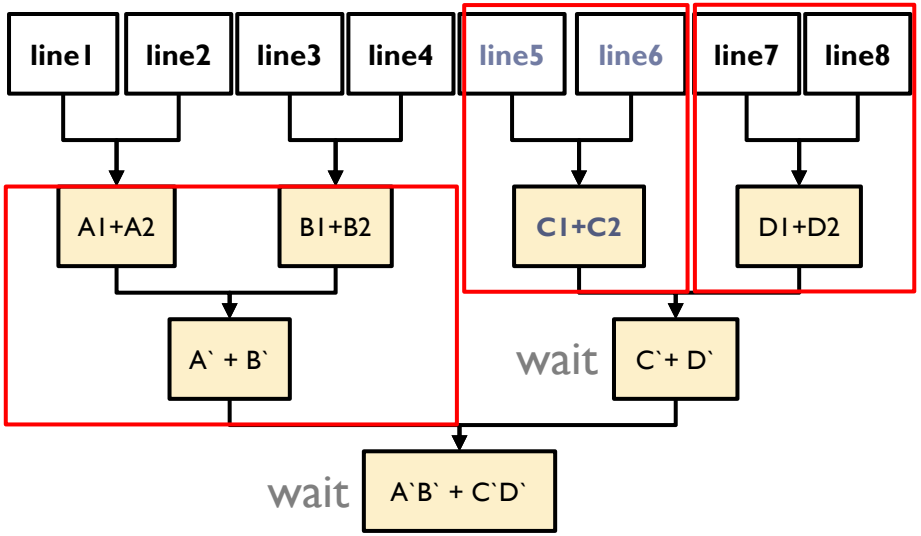
temp.txt

```
1
2
3
4
5
6
7
8
3
7
```

Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



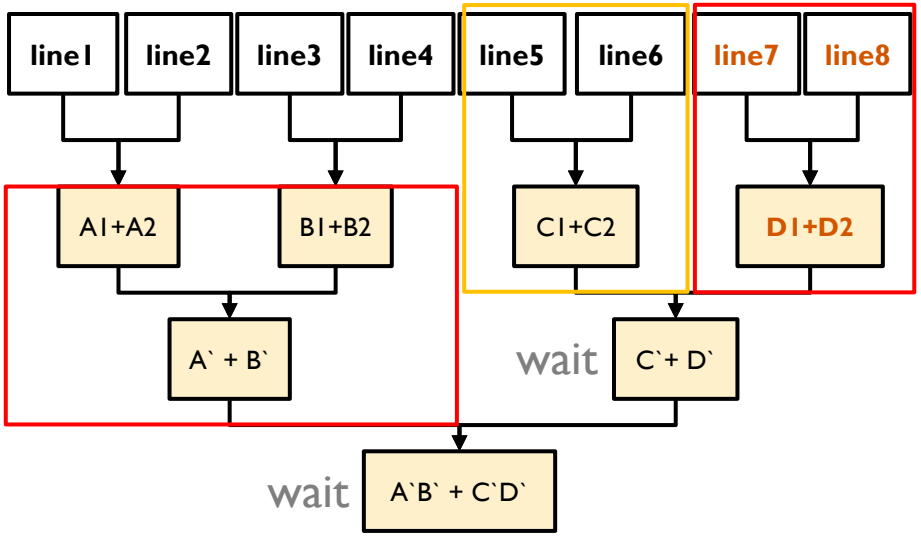
temp.txt

```
1
2
3
4
5
6
7
8
3
7
11
```

Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



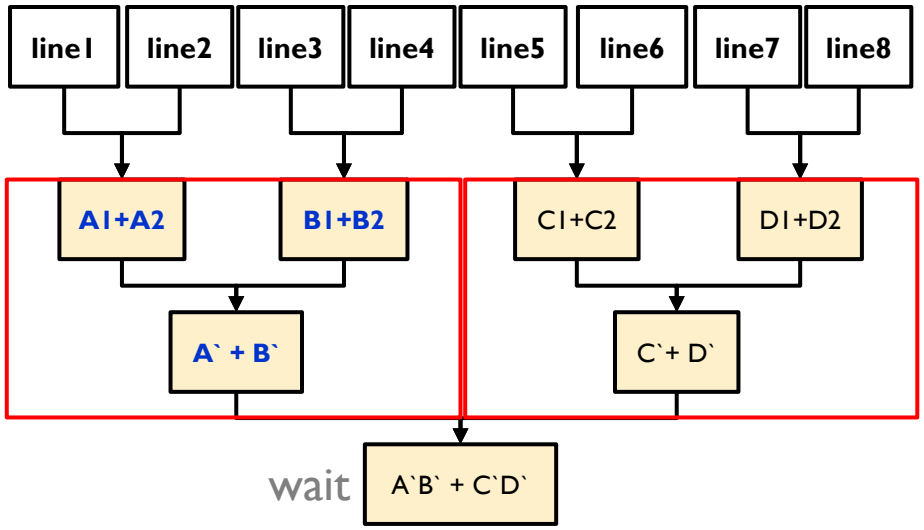
temp.txt

1 2 3 4 5 6 7 8
3 7 11 15

Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



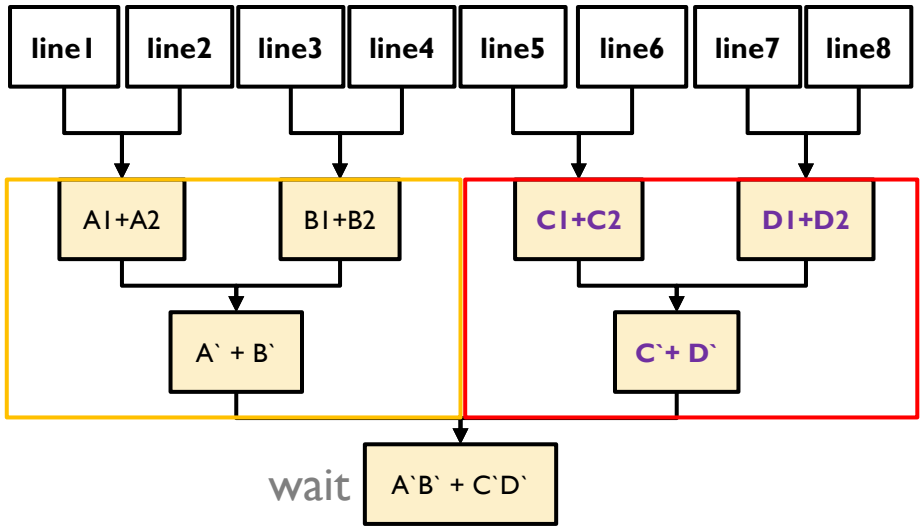
temp.txt

1 2 3 4 5 6 7 8
3 **7** 11 15 **10**

Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



temp.txt

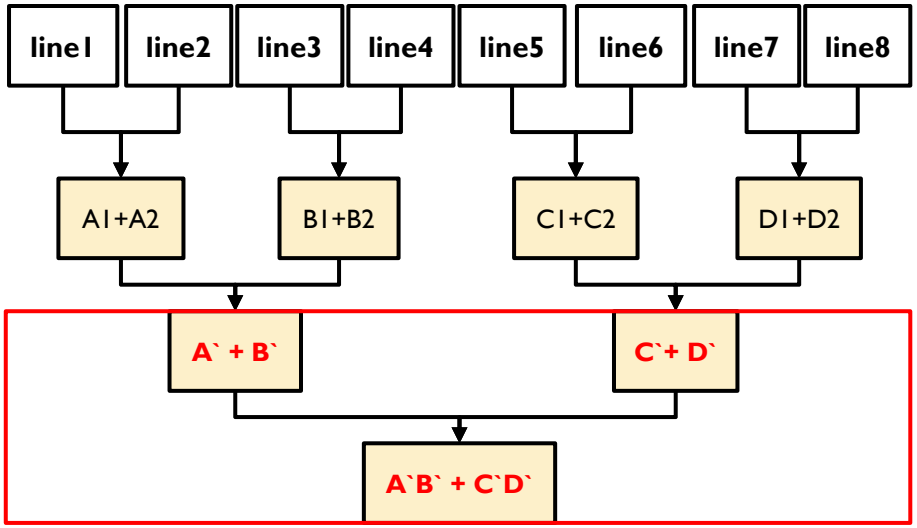
```
1
2
3
4
5
6
7
8
```

```
3
7
11
15
10
26
```

Assignment 3-2

- **Requirements**
  - 메인 프로세스와는 별개로 각각의 연산을 수행하는 프로세스 or 스레드 생성
    - 하위 depth의 프로세스는 새로 생성하거나, 상위 depth의 프로세스 1개를 이용

- **Hint**



temp.txt

1 2 3 4 5 6 7 8
3 7 11 15 **10** **26** **36**

Assignment 3-2

- **주의 사항**
  - 매 실험 전에 아래의 명령어를 수행할 것
    - 캐시 및 버퍼를 비워서 실험에 영향을 주는 요소를 제거

> - **rm –rf tmp***
> - $ **sync**
>   - Linux command to flush file system buffer
> - $ **echo 3 | sudo tee /proc/sys/vm/drop_caches**
>   - Linux commands to free pagecache, dentries, and inodes

  - 자식프로세스에서 부모프로세스로 값을 넘겨줄 때 (**exit ()**)

반환 값은 2^8 이상이면 안되며, 8-bit 만큼 right shift 해주어야 child process가 반환된 값을 정상적으로 확인할 수 있음. (e.g. a >> 8)

→ 이유를 보고서에 작성

Assignment 3-3

- **CPU scheduling simulator 제작**
  - Write a C program that implements **a simulator using different scheduling algorithms.**
  - The simulator will select tasks from the ready queue based on the scheduling algorithm.
  - There is no need for actual process creation or execution.
  - Print out the task selected to run at each time step in a format similar to a **Gantt chart**.

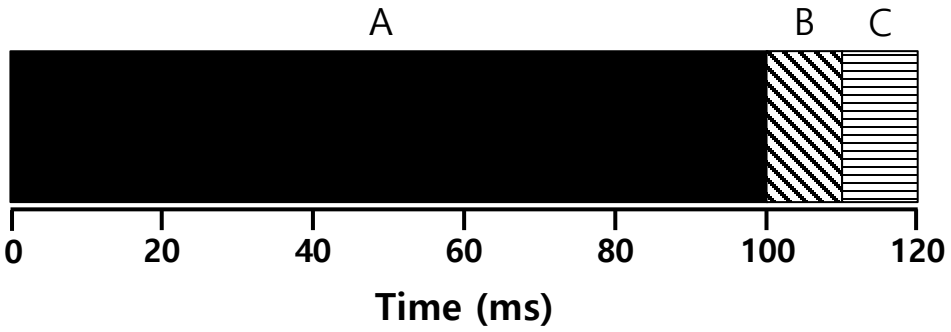


Fig. 1 example of Gantt chart

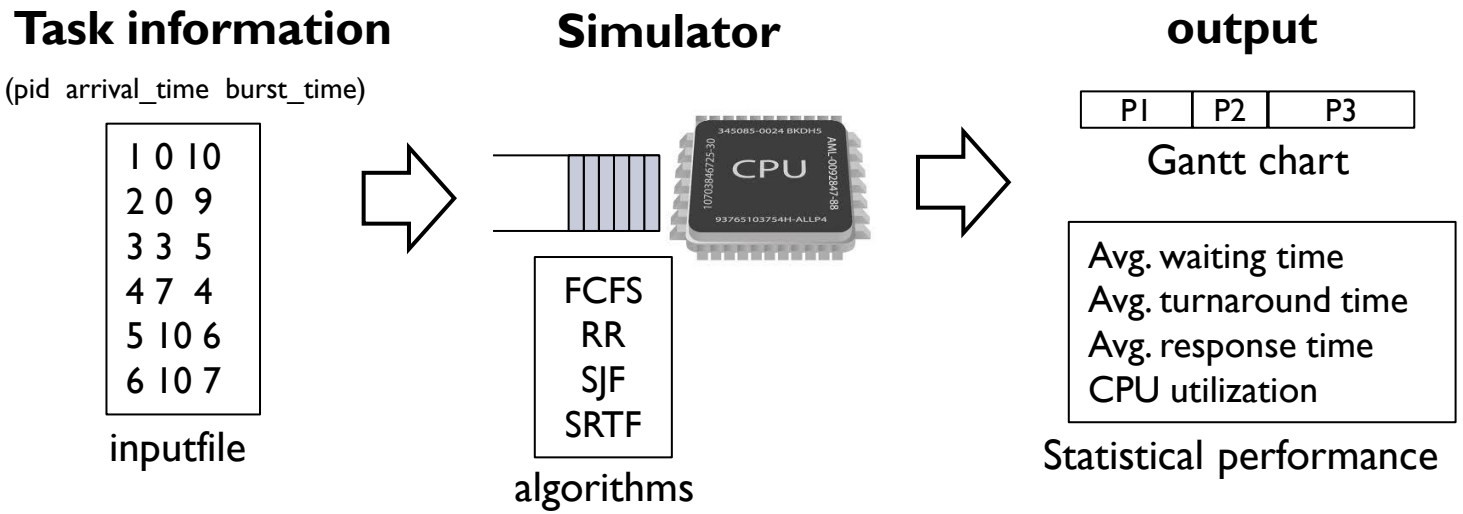
Project Description

- **The scheduling algorithms to be implemented**
  - First Come First Served (**FCFS**)
  - Round Robin (**RR**)
  - Shortest Job First (**SJF**)
  - Shortest Remaining Time First (**SRTF**)
    - also called Shortest Time-to-Completion First (STCF) or Preemptive Shortest Job First (PSJF)

- **FCFS and SJF are non-preemptive, while RR and SRTF are preemptive.**

Project Description

- **CPU scheduler simulator overview**



Project requirements

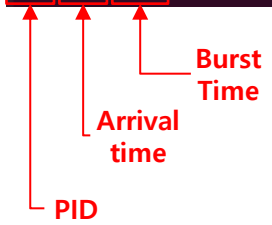
- **1. Implement the scheduling algorithms**
  - FCFS, RR (with time_quantum), SJF, and SRTF.
- **2. The program should print a Gantt chart showing task execution at each time step (in milliseconds).**
  - E.g. | P1 | P1 | P2 | P2 | P3 | P3 | P1 | P1 | P2|
- **3. After all tasks are completed, program must calculate and print**
  - Avg. waiting time, avg. response time, avg. turnaround time, and CPU utilization.
- **4. Context switching overhead is 0.1 milliseconds,**
  - and it should be factored into the simulation.

- **Note: If you use a static array to implement the ready queue, assume the maximum queue length is 1,000.**

Input

- **Task Information**
  - will be read from an input file.
  - The format for each task is:
    - **"pid arrival_time burst_time"**
      - **pid** is a unique process ID (integer).
      - **arrival_time** is the time the task arrives (in milliseconds).
      - **burst_time** is the CPU time requested by the task (in milliseconds).
    - Note that all time values are in milliseconds.
    - Input file example (input.1)

```
os2024123456@ubuntu:~/assgin3/assignment3/3-3$ cat input.1
1 0 10
2 0 9
3 3 5
4 7 4
5 10 6
6 10 7
```



Simulator

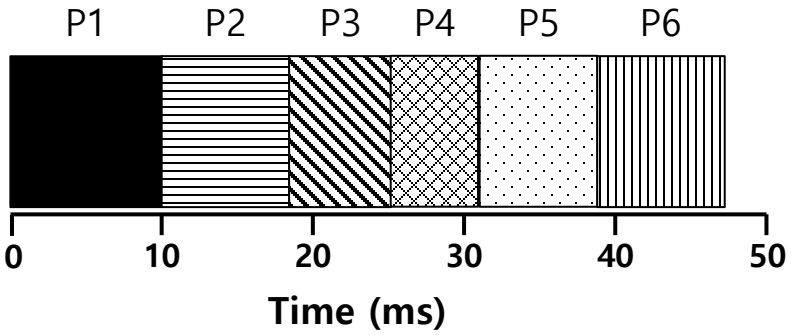
- **Command-line Usage**
  - "cpu_simulator input_file [FCFS|RR|SJF|SRTF] [time_quantum]"
    - **input_file** is the file containing task information.
    - The available scheduling algorithms are **FCFS, RR, SJF, and SRTF**.
    - The **time_quantum** parameter is only applicable to the RR algorithm.
      - The time_quantum must be specified for RR.

  - Examples
    - **cpu_simulator input.1 FCFS**
      - Simulate FCFS scheduling using the data file "input.1".
    - **cpu_simulator input.1 RR 2**
      - Simulate RR scheduling with a time quantum of 2 milliseconds using the data file "input.1".
    - **cpu_simulator input.1 SRTF**
      - Simulate Shortest Remaining Time First scheduling using the data file "input.1".

Output

- **Sample output(FCFS)**

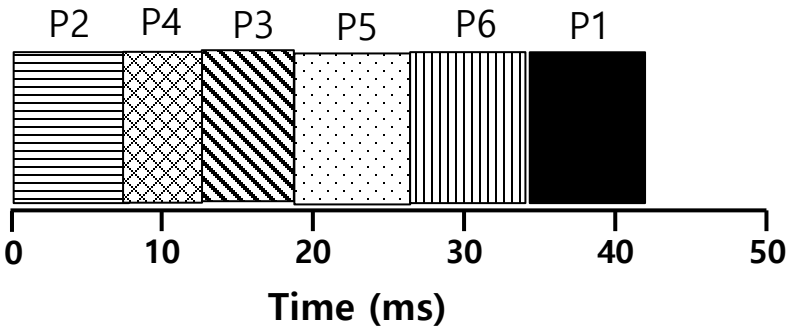
```
os2024123456@ubuntu:~/os/sched$ ./cpu_scheduler   input.1   FCFS
Gantt Chart:
| P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P2 | P2 | P2 | P2 | P2 | P2 | P2 | P2 | P2 |P3 | P3 | P3 | P3 | P3 |
 P4 | P4 | P4 | P4 | P5 | P5 | P5 | P5 | P5 | P5 | P6 | P6 | P6 | P6 | P6 | P6 | P6 |
Average Waiting Time = 14.17
Average Turnaround Time = 21.00
Average Response Time = 14.17
CPU Utilization = 98.56%
```



Output(Cont'd)

- **Sample Output(SJF)**

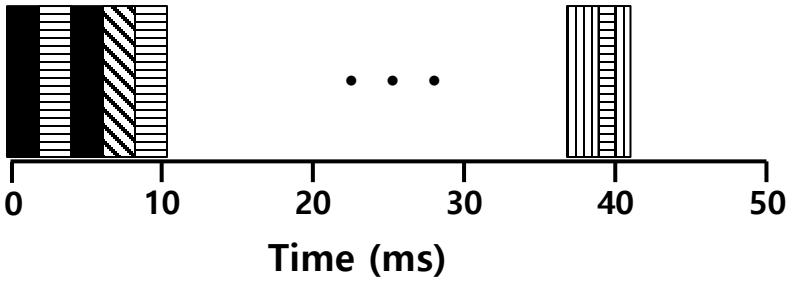
```
os2024123456@ubuntu:~/os/sched$ ./cpu_scheduler   input.1   SJF
Gantt Chart:
| P2 | P2 | P2 | P2 | P2 | P2 | P2 | P2 | P2 | P4 | P4 | P4 | P4 | P3 | P3 | P3 | P3 | P3 | P5 | P5 | P5 | P5 | P5 | P5 |
P6 | P6 | P6 | P6 | P6 | P6 | P6 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 |
Average Waiting Time = 10.83
Average Turnaround Time = 17.67
Average Response Time = 10.83
CPU Utilization = 98.56%
```



Output(Cont'd)

- **Sample output(RR)**

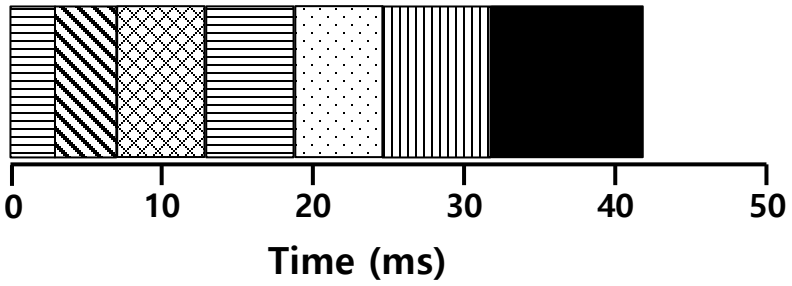
```
os2024123456@ubuntu:~/os/sched$ ./cpu_scheduler   input.1   RR 2
Gantt Chart:
| P1 | P1 | P2 | P2 | P1 | P1 | P3 | P3 | P2 | P2 | P1 | P1 | P4 | P4 | P3 | P3 | P5 | P5 | P6 | P6 | P2 | P2 | P1 | P1 |
 P4 | P4 | P3 | P5 | P5 | P6 | P6 | P2 | P2 | P1 | P1 | P5 | P5 | P6 | P6 | P2 | P6 |
Average Waiting Time = 22.5
Average Turnaround Time = 29.3
Average Response Time = 4.00
CPU Utilization = 94.91%
```



Output(Cont'd)

- **Sample Output(SRTF)**

```
os2024123456@ubuntu:~/os/sched$ ./cpu_scheduler   input.1   SRTF
Gantt Chart:
| P2 | P2 | P2 | P3 | P3 | P3 | P3 | P3 | P4 | P4 | P4 | P4 | P2 | P2 | P2 | P2 | P2 | P2 | P5 | P5 | P5 | P5 | P5 | P5 |
 P6 | P6 | P6 | P6 | P6 | P6 | P6 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 |
Average Waiting Time = 10.50
Average Turnaround Time = 17.33
Average Response Time = 9.0
CPU Utilization = 98.32%
```



Report Requirements

- **Ubuntu 20.04.6 Desktop 64bits 환경에서 채점**
- **Copy 발견 시 0점 처리**
- **보고서 구성**
  - **보고서 표지**
    - 수업 명, 과제 이름, 담당 교수님, 학번, 이름 필히 명시
      - 과제 이름 ➔ Assignment #1
  - **과제 내용**
    - Introduction
      - 과제 소개 - 4줄 이상(background 제외) 작성
    - Result
      - 수행한 내용을 캡처 및 설명
      - **Assignment 3-3 : 보고서에 각 알고리즘 별 성능 결과를 비교 분석**
    - 고찰
      - 과제를 수행하면서 느낀 점 작성
    - Reference
      - 과제를 수행하면서 참고한 내용을 구체적으로 기록
      - 강의자료만 이용한 경우 생략 가능

Report Requirements

- **Source**
  - Assignment 3-1(3개)
    - Process_tracer.c
    - Makefile
    - Fork() 호출 횟수를 count 하기 위해 수정한 커널 소스코드
  - Assigment 3-2(4개)
    - numgen.c / fork.c / thread.c / Makefile
  - Assigment 3-3(2개)
    - cpu_scheduler.c / Makefile

  - 각 코드 파일은 디렉토리를 나누어서 제출
    - E.g)
      - Assignment2-1/os_ftrace.c
      - Assignment2-2/os_ftracehooking.c
      - Assignment2-3/ ftracehooking.c
        / ftarcehooking.h
        / iotracehooking.c
        / Makefile

- **Copy 발견 시 0점 처리**

Report Requirements

- **Softcopy Upload**
  - 제출 파일
    - 보고서 + 소스파일 **[하나의 압축 파일로 압축하여 제출(tar.xz)]**
    - 보고서(.pdf. 파일 변환)
    - 소스코드(**Comment 반드시 포함**)

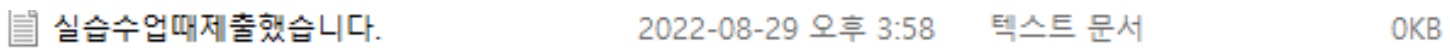
  - 보고서 및 압축 파일 명 양식
  - **OS_Assignment1 _*수강분류코드_학번_이름* 으로 작성**

수강요일	이론1 월6수5	이론2 목3	실습 금56
수강분류코드	A	B	C

    - **예시 #1)–이론(월6수5)만 수강하는 학생인 경우**
      - **보고서 OS_Assignment1_A_2024123456_홍길동.pdf**
      - **압축 파일 명: OS_Assignment1_A_2024123456_홍길동.tar.xz**
    - **예시 #2)–이론(월6수5 or 목3)과 실습 모두 수강하는 학생인 경우**
      - **보고서 OS_Assignment1_C_2024123456_홍길동.pdf**
      - **압축 파일 명: OS_Assignment1_C_2024123456_홍길동.tar.xz**
      - **+ "해당 이론반 txt 파일 제출"**

Report Requirements

- **실습 수업을 수강하는 학생인 경우**
  - **실습 과목에 과제를 제출**(.tar.xz)
  - **이론 과목에 간단한 .txt 파일**로 제출



    - **이론 과목에 .txt 파일 미 제출 시 감점**
    - **.tar.xz 파일로 제출 하지 않을 시 감점**

- **과제 제출**
  - **KLAS – 강의 과제 제출**
  - **2024년 11월 7일 목요일 23:59까지 제출**
    - **딜레이 받지 않음**
      - 제출 마감 시간 내 미제출시 해당 과제 **0점 처리(예외 없음)**

KWANGWOON UNIVERSITY

Appendix

System Software Laboratory
School of Computer and Information Engineering
Kwangwoon Univ.

CPU utilization

- **CPU Utilization**
  - the ratio of the time the CPU is actively working on tasks to the total simulation time.
  - It helps measure how efficiently the CPU is being utilized.

- **How to Calculate CPU Utilization**
  - $\text{CPU Utilization} = \frac{\text{Total CPU Busy Time}}{\text{Total Time}} * 100$
  - Total CPU Busy Time
    - The total time the CPU spends executing tasks
    - i.e. the sum of the burst times of all processes
  - Total Time
    - The total time elapsed during the simulation,
    - includes time spent on task execution, idle time, and context switching overhead.

CPU utilization example

- **Assuming that processes 1, 2, and 3 were scheduled with FCFS.**
  - Process 1: 10 seconds burst time
  - Process 2: 9 seconds burst time
  - Process 3: 5 seconds burst time
  - Context switch overhead is 0.1 second per switch.
- **Answer**
  - Total CPU Busy Time
    - 10 + 9 + 5 = 24 seconds
  - Total Time
    - 3 context switches, so the total context switching overhead is 0.1 × 3 = 0.3 seconds.
    - the total time is 24 + 0.3 = 24.3 seconds.
  - CPU Utilization
    - $\frac{24}{24.3} * 100 = 98.77\%$