OS Term Project Report

Shell commands

```
cd TermProject
g++ -std=c++17 "term_project_rr.cpp" -o "term_project_rr" && "./term_project_rr"
```

Rubric Table

Whether the code implementation is complete	**O**
The setting of time quantum (which executed without errors)	**3s (default)**
Whether IPC message queue is used	**O**
How the i/o operation is implemented	**option 2**
Whether the multi-level queue was implemented	**X**

Implementations

1. Initialization

Headers

```
#include <iostream>
#include <fstream>
#include <vector>
#include <queue>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/time.h>
#include <cstdlib>
#include <ctime>
#include <cstring>
#define READY 2      // 준비상태
#define RUNNIG 3     // 실행상태
#define WAITING 4    // 대기상태(I/O 발생)
#define TERMINATED 5 // 종료상태
```

- 로그 파일 기록을 위해 `fstream` include

- `ALARM` 시그널 생성을 위해 `signal.h` include
- 메세지 큐를 이용한 IPC 통신을 위해 `sys/ipc.h`, `sys/msg.h` include
- Ready queue와 I/O queue 구현을 위해 `queue` include
- 그 외 각종 기능(대기, 문자열 변환 등)을 위한 헤더 include
- 프로세스의 상태를 매크로로 정의

PCB Structure

```
struct PCB
{
    int idx;                // Work의 index 번호 (0~9)
    int pid;                // Child process id
    string processName;     // Work0 ~ Work9
    int state;              // Process state (READY, RUNNIG, WAITING, TERMINATED)
    int remainingCPUBurst;  // 남은 CPU burst
    int remainingIoBurst;   // 남은 I/O burst
    int ioStartTimeQuantum; // I/O가 시작될 time quantum
    int ioStartsIn;    // I/O 시작까지 남은 시간
};
```

- 프로세스(Work)에 대한 정보를 기록하기 위한 PCB 정의
- 문맥교환 및 기록을 위해 다양한 값을 저장한다.

IPC Communication (with Message Queue)

```
struct msgBuffer
{
    long mtype = 0; // Work의 pid나 work의 상태
    PCB process;
};

void sendMsg(int type, PCB process)
{
    msgBuffer msg;
    msg.mtype = type;
    msg.process = process;
    if (msgsnd(msgQueueID, &msg, sizeof(PCB), 0) == -1)
    {
        perror("msgsnd");
        exit(1);
    }
}
```

- `msgBuffer` 의 `mtype` 에는 Work의 pid나 상태가 들어간다.
  - pid가 들어간 경우엔, 부모 프로세스가 자식 프로세스에게 메세지를 전달할 때이다.
  - Work의 상태 (ex. `TERMINATED`)가 들어간 경우, 자식 프로세스가 부모 프로세스에게 자신의 상태를 전달해야 하는 상황이다.
- `sendMsg()` 함수는 `mtype` 을 인자로 받아 부모와 자식 프로세스 모두 사용 가능하도록 구현하였다.

Global Variables

```
// Global variables
int msgQueueID;
const int childNum = 10;
int responseTimes[childNum] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}; // 응답시간
int turnaroundTimes[childNum];                                           // 반환시간
int totalCpuTime = 0;
int totalIoTime = 0;

// Process
int pid;               // process id (0이면 자식 프로세스)
PCB currentProcess;    // 현재 실행중인 프로세스
queue<PCB> readyQueue; // ready queue
PCB ioProcess;         // 현재 I/O 작업중인 프로세스
queue<PCB> ioQueue;    // I/O queue

// Time
int currentTime = 0;             // 현재 시간
const int timeQuantum = 3;       // time quantum (변경 가능)
const int timerTickInterval = 1; // Timer tick interval in seconds
```

- Ready queue와 I/O queue 구현은 queue STL 사용
- 가장 작은 실행 단위는 1초로 정의

Initialize

```
void init()
{
    srand(time(NULL)); // random seed

    sigset(SIGALRM, handleAlarmSignal); // SIGALRM 신호 오면 해당 콜백함수 실행

    // Create the message queue
    key_t key = ftok(".", 1);
    if ((msgQueueID = msgget(key, 0666 | IPC_CREAT)) == -1) exit(1);
    if (msgctl(msgQueueID, IPC_RMID, NULL) == -1) exit(1); // delete msg queue
    if ((msgQueueID = msgget(key, 0666 | IPC_CREAT | IPC_EXCL)) == -1) exit(1);
```

```
    // Initialize child processes and their PCBs
    createChildProcesses();
}
```

- CPU burst time과 I/O burst time의 랜덤한 생성을 위해 `srand()` 실행
- `SIGALRM` 신호가 발생하면 `handleAlarmSignal()` 콜백 함수를 실행하도록 `sigset()` 으로 설정
- 메세지 큐 생성 시, 동일한 ID의 메세지 큐가 이미 존재한다면 해당 큐를 삭제하고 새로 만든다.
- `createChildProcesses()` 를 통해 10개의 자식 프로세스가 생성된다.

2. Child Process (Work)

Create Child Process

```
void createChildProcesses()
{

    for (int i = 0; i < childNum; ++i)
    {
        pid = fork();
        if (pid < 0)
        {
            printf("Error in fork()\n");
            exit(-1);
        }

        PCB newProcess;
        newProcess.idx = i;                                 // Process 번호
        newProcess.pid = pid;                               // 자식의 pid 저장
        newProcess.processName = "Work" + to_string(i);     // Process 이름
        newProcess.state = READY;                           // Process 상태
        newProcess.remainingCPUBurst = (rand() % 10) + 1; // Random CPU burst 생성
        totalCpuTime += newProcess.remainingCPUBurst;

        if (pid == 0) // Child process
        {
            childProcess();
            cout << "Work" << i << " end" << endl;
            exit(0);
        }
        else // Parent process
            readyQueue.push(newProcess);
    }
}
```

- `createChildProcesses()` 를 통해 10개의 자식 프로세스가 생성되고, 각 프로세스의 idx, pid, 이름, 상태가 설정된다. 또한 CPU burst time이 1~10 사이의 정수로 랜덤하게 생성된다.
- 자식 프로세스는 `fork()` 로 부모 프로세스에서 분기되고, `childProcess()` 함수로 진입하여 CPU burst와 I/O burst를 무한히 실행하게 된다.
  - 부모 프로세스는 `fork()` 시, `pid` 변수에 자식의 pid가 들어가게 되고, 해당 프로세스를 `readyQueue` 에 넣는다.
- 자식 프로세스의 CPU 및 I/O 작업이 모두 종료되면 해당 프로세스를 종료시킨다.

Proceeding Child Process

```
void childProcess()
{
    while (1)
    {
        msgBuffer msg;
        // 자식의 pid와 같은 mtype의 메세지 올때까지 기다림
        if (msgrcv(msgQueueID, &msg, sizeof(PCB), getpid(), 0) == -1)
        {
            perror("msgrcv");
            exit(1);
        }
        // 메세지 수신 성공시 CPU 할당 -> time quantum 할당
        PCB process = msg.process;
        int remainingTimeQuantum = timeQuantum;

        while (remainingTimeQuantum)
        {
            cout << printProcess(process);

            if (process.ioStartTimeQuantum == remainingTimeQuantum) // I/O 시작시간 도달

                process.state = WAITING;
                process.ioStartsIn = 0;
                process.ioStartTimeQuantum = 0;
                sendMsg(WAITING, process); // CPU 할당 해제시기키 위해 부모에게 메세지 전송
                break;                     // CPU 할당 해제
            }

            sleep(timerTickInterval); // 1초

            process.remainingCPUBurst--;
            remainingTimeQuantum--;

            if (process.remainingCPUBurst == 0) // CPU burst 완료
            {
                process.state = TERMINATED;
                sendMsg(TERMINATED, process); // send msg to parent process
                return;
            }
        }
```

```
    }
}
```

- `childProcess()` 에 진입하면, `msgrcv()` 에 의해 부모에게 메세지를 받기 전까지 block 된다.
- 메세지 수신 성공 시 CPU를 할당받은 것으로 가정하고 time quantum을 부여한다.
- 남은 time quantum 만큼 CPU 작업을 수행하며 남은 CPU burst 시간과 time quantum을 1초씩 줄인다.
- 만약 I/O 시작 시간에 도달하면, 부모 프로세스에게 대기상태가 되었다는 메세지를 전송하고 CPU 할당을 해제한다.
- CPU burst가 완료되면, 부모 프로세스에게 종료상태가 되었다는 메세지를 전송하고 해당 자식 프로세스를 종료한다.

3. Parent Process (Round Robin Scheduler)

Round Robin Scheduling

```
void roundRobin()
{
    int remainingTimeQuantum = 0;
    while (currentProcess.remainingCPUBurst > 0 || ioProcess.remainingIoBurst > 0
            || !readyQueue.empty() || !ioQueue.empty())
    {
        msgBuffer msg;
        if (remainingTimeQuantum == 0) // time quantum 소진시 스케줄링
        {
            raise(SIGALRM); // SIGALRM 신호 발생 -> 자식 프로세스 실행
            remainingTimeQuantum = timeQuantum;
        }

        writeLog("[Time: start of " + to_string(currentTime) + " sec]\n");
        sleep(timerTickInterval); // 1초
        currentProcess.remainingCPUBurst--; ioProcess.remainingIoBurst--;
        currentProcess.ioStartsIn--; remainingTimeQuantum--;

        writeLog("[Time: end of " + to_string(currentTime) + " sec]\n");
        // 자식 프로세스 종료 메세지 수신
        if (msgrcv(msgQueueID, &msg, sizeof(PCB), TERMINATED, IPC_NOWAIT) != -1)
        {
            turnaroundTimes[msg.process.idx] = currentTime + 1;
            if (currentProcess.pid == msg.process.pid) // 메세지 늦게 온 경우 고려
            {
                currentProcess = PCB(); // 현재 프로세스를 cpu에서 해제
                remainingTimeQuantum = 0;
            }
        }
        // io 발생시 (time quantum 소진과 동시에 일어나지 않음)
        if (msgrcv(msgQueueID, &msg, sizeof(PCB), WAITING, IPC_NOWAIT) != -1)
```

```
        {
            ioQueue.push(msg.process); // 프로세스를 io queue에 넣음
            currentProcess = PCB();    // 현재 프로세스를 cpu에서 해제
            remainingTimeQuantum = 0;
        }

        // io 스케줄링(FCFS)
        ioScheduling(remainingTimeQuantum);
        currentTime++;
    }

    writeLog("[Time: " + to_string(currentTime) + " sec] - end of simulation\n");
    return;
}
```

- 부모 프로세스는 round robin 스케줄링 알고리즘을 사용하여 자식 프로세스들을 스케줄링 한다.
- 해당 스케줄링은 현재 작업중인 CPU 혹은 I/O 작업이 있거나, ready queue나 I/O queue가 비어있지 않으면 계속 진행한다.
- 만약 현재 남은 time quantum을 모두 소진하면, `SIGALRM` 신호를 발생시켜 `handleAlarmSignal()` 를 호출한다. 함수가 호출되면, 해당 자식 프로세스에게 CPU를 할당하도록 메세지를 보낸다.
- 단위 시간(1초)이 흐른 후, 부모 프로세스는 자식 프로세스로부터 받은 메세지가 있는지 확인한다.
    - 종료 메세지를 받으면, 현재 실행중인 프로세스를 삭제한다. 이때 메세지가 늦게 온 경우를 고려하여, 현재 실행중인 프로세스가 메세지로 받은 프로세스와 일치하는 경우에만 삭제한다.
    - I/O가 발생 메세지를 받으면, 해당 프로세스를 I/O queue에 넣고 현재 프로세스를 CPU 해제한다.
- 만약 I/O 작업이 있다면, `ioScheduling()` 함수를 실행하여 스케줄링 한다.

Child Process CPU Allocation (with ALARM Signal)

```
void handleAlarmSignal(int signum)
{
    if (currentProcess.remainingCPUBurst > 0) // 만약 CPU burst가 남아있으면 ready queue에 넣음
        readyQueue.push(currentProcess);

    if (!readyQueue.empty()) // 메세지 큐에 타임슬라이스 보내기
    {
        PCB nextProcess = readyQueue.front();
        readyQueue.pop();
        nextProcess.state = RUNNIG;
        if (responseTimes[nextProcess.idx] == -1)
```

```
            responseTimes[nextProcess.idx] = currentTime;

        // IO 발생 여부
        if (nextProcess.remainingCPUBurst >= timeQuantum && rand() % 2 == 0) // 50% 확률로 I/O
        {
            // io start time radomly determined
            nextProcess.ioStartTimeQuantum = rand() % (timeQuantum - 1) + 1;
            nextProcess.ioStartsIn = timeQuantum - nextProcess.ioStartTimeQuantum;
            // io duration randomly determined
            nextProcess.remainingIoBurst = (rand() % 10) + 1;
            totalIoTime += nextProcess.remainingIoBurst;
        }

        // Dispatch the next process (for example, by sending an IPC message)
        currentProcess = nextProcess;
        sendMsg(nextProcess.pid, nextProcess); // send msg to child process
    }
}
```

- `SIGALRM` 신호가 발생하면, 해당 자식 프로세스에게 CPU를 할당하도록 메세지를 보낸다.
- 새 작업을 수행하기 전에 현재 수행하고 있는 작업의 CPU burst가 남아있으면 ready queue에 넣는다.
- Ready queue가 비어있지 않다면, 가장 앞에 있는 새 작업을 꺼내 dispatch 한다. 새 작업을 꺼낼 때, 다음 몇가지 사항을 수정하여 작업을 실행한다.
  - 새 작업이 실행중인 것을 나타내기 위해 상태를 `RUNNIG` 으로 설정한다.
  - Option 2에 명시된 바와 같이, 자식 프로세스가 실행되면 I/O수행여부, I/O지속시간, I/O시작시간이 랜덤하게 결정된다.
    - I/O 발생은 현재 남은 CPU burst가 time quantum보다 많을 때만 발생하도록 하였다. 작업이 끝나면서 I/O 작업을 수행하는 것이 부자연스럽다고 판단하여, 해당 프로젝트에서는 작업이 끝날 때 I/O가 발생하지 않도록 구현하였다.
    - I/O는 50% 확률로 발생하며, I/O 지속시간은 1~10초로 랜덤하게 발생한다.
- 현재 작업중인 프로세스를 update 하고, 자식 프로세스에게 메세지를 CPU를 할당한다. 메세지를 보낼 때는 PCB에 저장된 자식 프로세스의 pid를 참조하여 보내고, 자식 프로세스가 메세지를 받으면 block이 해제된다.

I/O Scheduling

```
void ioScheduling(int &remainingTimeQuantum)
{
    if (ioProcess.remainingIoBurst <= 0) // 수행중인 io 작업 없거나, 완료시
    {
```

```
        if (ioProcess.state == WAITING) // 대기상태 였는데 완료됨
        {
            if (ioProcess.remainingCPUBurst > 0) // 만약 CPU burst가 남아있으면 ready queue에 넣음
            {
                ioProcess.state = READY;
                readyQueue.push(ioProcess);
            }
        }

        ioProcess = PCB();    // 현재 실행중인 io 작업 없음
        if (!ioQueue.empty()) // io queue에 프로세스가 있으면 즉시 dispatch
        {
            ioProcess = ioQueue.front();
            ioQueue.pop();
        }

        // CPU 작업을 dispatch 가능하다면
        if (currentProcess.remainingCPUBurst <= 0 && !readyQueue.empty())
            remainingTimeQuantum = 0; // 즉시 dispatch 할 수 있도록 time quantum 초기화
    }
}
```

- I/O 작업은 중간을 중간에 끊었다가 다시 실행하는 것은 부자연스러우므로 I/O 스케줄링은 FCFS 알고리즘으로 수행한다.
- 현재 수행중인 I/O 작업이 없거나 완료되면 스케줄링을 수행한다.
- 현재 수행중인 I/O 작업이 완료된 상황이라면 해당 프로세스를 `READY` 상태로 바꾸고, 이를 ready queue에 넣는다.
- 만약 I/O queue에 프로세스가 있으면 즉시 dispatch 한다.
- 만약 현재 CPU를 할당받은 프로세스가 없고 ready queue로부터 dispatch 가능하다면, 즉시 dispatch 할 수 있도록 time quantum 초기화한다.

4. Experimental results

Different Time Quantums

- Time quantum: 2

```
[Time: 117 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 110 sec
Work1 response time: 1 sec, turnaround time: 55 sec
Work2 response time: 3 sec, turnaround time: 17 sec
Work3 response time: 5 sec, turnaround time: 6 sec
Work4 response time: 6 sec, turnaround time: 117 sec
Work5 response time: 7 sec, turnaround time: 45 sec
Work6 response time: 8 sec, turnaround time: 40 sec
Work7 response time: 9 sec, turnaround time: 106 sec
Work8 response time: 11 sec, turnaround time: 70 sec
```

```
Work9 response time: 12 sec, turnaround time: 68 sec
Average response time: 6.200000 sec, average turnaround time: 63.400000 sec
Throughput: 0.015773 jobs/sec
CPU utilization: 53.846154%
I/O utilization: 97.435897%
```

- Time quantum: 3

```
[Time: 81 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 39 sec
Work1 response time: 1 sec, turnaround time: 73 sec
Work2 response time: 3 sec, turnaround time: 21 sec
Work3 response time: 6 sec, turnaround time: 72 sec
Work4 response time: 7 sec, turnaround time: 61 sec
Work5 response time: 10 sec, turnaround time: 81 sec
Work6 response time: 11 sec, turnaround time: 27 sec
Work7 response time: 14 sec, turnaround time: 45 sec
Work8 response time: 15 sec, turnaround time: 31 sec
Work9 response time: 18 sec, turnaround time: 57 sec
Average response time: 8.500000 sec, average turnaround time: 50.700000 sec
Throughput: 0.019724 jobs/sec
CPU utilization: 70.370370%
I/O utilization: 93.827160%
```

- Time quantum: 4

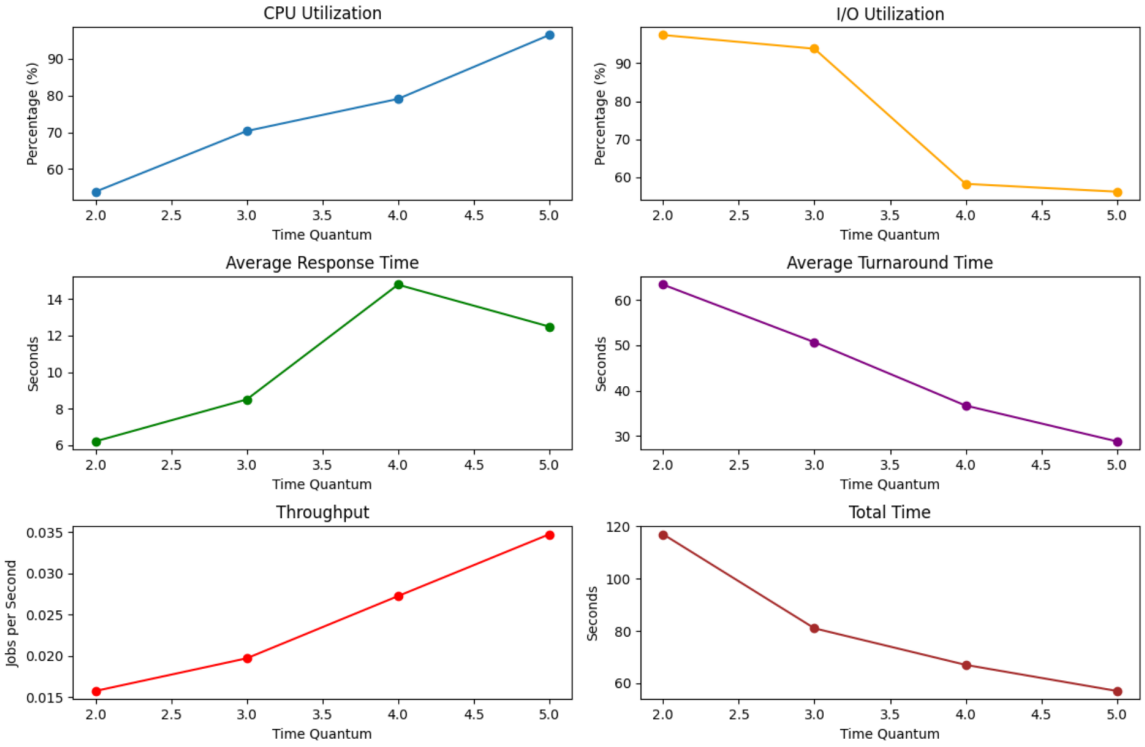
```
[Time: 67 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 50 sec
Work1 response time: 4 sec, turnaround time: 5 sec
Work2 response time: 5 sec, turnaround time: 67 sec
Work3 response time: 9 sec, turnaround time: 38 sec
Work4 response time: 13 sec, turnaround time: 17 sec
Work5 response time: 17 sec, turnaround time: 43 sec
Work6 response time: 20 sec, turnaround time: 42 sec
Work7 response time: 24 sec, turnaround time: 27 sec
Work8 response time: 27 sec, turnaround time: 46 sec
Work9 response time: 29 sec, turnaround time: 32 sec
Average response time: 14.800000 sec, average turnaround time: 36.700000 sec
Throughput: 0.027248 jobs/sec
CPU utilization: 79.104478%
I/O utilization: 58.208955%
```

- Time quantum: 5

```
[Time: 57 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 36 sec
Work1 response time: 5 sec, turnaround time: 6 sec
Work2 response time: 6 sec, turnaround time: 8 sec
Work3 response time: 8 sec, turnaround time: 10 sec
Work4 response time: 10 sec, turnaround time: 14 sec
Work5 response time: 14 sec, turnaround time: 52 sec
```

```
Work6 response time: 15 sec, turnaround time: 57 sec
Work7 response time: 18 sec, turnaround time: 46 sec
Work8 response time: 22 sec, turnaround time: 27 sec
Work9 response time: 27 sec, turnaround time: 32 sec
Average response time: 12.500000 sec, average turnaround time: 28.800000 sec
Throughput: 0.034722 jobs/sec
CPU utilization: 96.491228%
I/O utilization: 56.140351%
```

- Result



  - 구현된 프로그램에서 CPU, I/O burst time이 랜덤으로 결정되기 때문에 매번 같은 형태가 나오진 않는다.
  - time quantum이 작은 경우 (잦은 문맥 교환)
    - CPU를 할당 받을 때 I/O 수행여부가 결정되므로, CPU 할당을 짧게 자주 받는 해당 경우에서 I/O가 많이 발생한다.
    - 문맥 교환이 잦기 때문에 평균 응답 시간은 빠르고, 평균 반환 시간은 늦어지는 경향이 있다.
    - 단위 시간동안 적은 양의 작업을 수행한다. (낮은 throughput)
    - 총 소요 시간이 긴 경향을 보인다.
  - time quantum이 큰 경우 (뜸한 문맥 교환)

- CPU를 할당 받을 때 I/O 수행여부가 결정되므로, CPU 할당을 길고 뜸하게 받는 해당 경우에서 I/O가 적게 발생한다.
- 문맥 교환이 뜸하기 때문에 평균 응답 시간은 늦고, 평균 반환 시간은 가장 빠른 경향이 있다.
- 단위 시간동안 많은 양의 작업을 수행한다. (높은 throughput)
- 총 소요 시간이 짧은 경향을 보인다.

Different I/O Occurence Rate

- I/O occurence rate: 0%

```
[Time: 60 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 2 sec
Work1 response time: 2 sec, turnaround time: 52 sec
Work2 response time: 5 sec, turnaround time: 53 sec
Work3 response time: 8 sec, turnaround time: 10 sec
Work4 response time: 10 sec, turnaround time: 54 sec
Work5 response time: 13 sec, turnaround time: 40 sec
Work6 response time: 16 sec, turnaround time: 55 sec
Work7 response time: 19 sec, turnaround time: 45 sec
Work8 response time: 22 sec, turnaround time: 0 sec
Work9 response time: 25 sec, turnaround time: 59 sec
Average response time: 12.000000 sec, average turnaround time: 37.000000 sec
Throughput: 0.027027 jobs/sec
CPU utilization: 100.000000%
I/O utilization: 0.000000%
```

- I/O occurence rate: 10%

```
[Time: 53 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 43 sec
Work1 response time: 3 sec, turnaround time: 29 sec
Work2 response time: 6 sec, turnaround time: 45 sec
Work3 response time: 9 sec, turnaround time: 10 sec
Work4 response time: 10 sec, turnaround time: 50 sec
Work5 response time: 13 sec, turnaround time: 53 sec
Work6 response time: 15 sec, turnaround time: 52 sec
Work7 response time: 18 sec, turnaround time: 39 sec
Work8 response time: 21 sec, turnaround time: 47 sec
Work9 response time: 23 sec, turnaround time: 25 sec
Average response time: 11.800000 sec, average turnaround time: 39.300000 sec
Throughput: 0.025445 jobs/sec
CPU utilization: 98.113208%
I/O utilization: 37.735849%
```

- I/O occurence rate: 50%

```
[Time: 81 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 39 sec
Work1 response time: 1 sec, turnaround time: 73 sec
Work2 response time: 3 sec, turnaround time: 21 sec
Work3 response time: 6 sec, turnaround time: 72 sec
Work4 response time: 7 sec, turnaround time: 61 sec
Work5 response time: 10 sec, turnaround time: 81 sec
Work6 response time: 11 sec, turnaround time: 27 sec
Work7 response time: 14 sec, turnaround time: 45 sec
Work8 response time: 15 sec, turnaround time: 31 sec
Work9 response time: 18 sec, turnaround time: 57 sec
Average response time: 8.500000 sec, average turnaround time: 50.700000 sec
Throughput: 0.019724 jobs/sec
CPU utilization: 70.370370%
I/O utilization: 93.827160%
```

- I/O occurence rate: 25%

```
[Time: 48 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 3 sec
Work1 response time: 3 sec, turnaround time: 37 sec
Work2 response time: 6 sec, turnaround time: 28 sec
Work3 response time: 9 sec, turnaround time: 48 sec
Work4 response time: 10 sec, turnaround time: 29 sec
Work5 response time: 13 sec, turnaround time: 14 sec
Work6 response time: 14 sec, turnaround time: 16 sec
Work7 response time: 16 sec, turnaround time: 18 sec
Work8 response time: 18 sec, turnaround time: 47 sec
Work9 response time: 21 sec, turnaround time: 22 sec
Average response time: 11.000000 sec, average turnaround time: 26.200000 sec
Throughput: 0.038168 jobs/sec
CPU utilization: 95.833333%
I/O utilization: 20.833333%
```

- I/O occurence rate: 33%

```
[Time: 63 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 62 sec
Work1 response time: 3 sec, turnaround time: 48 sec
Work2 response time: 4 sec, turnaround time: 27 sec
Work3 response time: 7 sec, turnaround time: 57 sec
Work4 response time: 8 sec, turnaround time: 40 sec
Work5 response time: 9 sec, turnaround time: 55 sec
Work6 response time: 12 sec, turnaround time: 53 sec
Work7 response time: 13 sec, turnaround time: 45 sec
Work8 response time: 14 sec, turnaround time: 30 sec
Work9 response time: 17 sec, turnaround time: 63 sec
Average response time: 8.700000 sec, average turnaround time: 48.000000 sec
Throughput: 0.020833 jobs/sec
CPU utilization: 100.000000%
I/O utilization: 85.714286%
```

- I/O occurence rate: 66%

```
[Time: 79 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 28 sec
Work1 response time: 2 sec, turnaround time: 79 sec
Work2 response time: 4 sec, turnaround time: 51 sec
Work3 response time: 7 sec, turnaround time: 8 sec
Work4 response time: 8 sec, turnaround time: 30 sec
Work5 response time: 11 sec, turnaround time: 13 sec
Work6 response time: 13 sec, turnaround time: 69 sec
Work7 response time: 15 sec, turnaround time: 41 sec
Work8 response time: 16 sec, turnaround time: 39 sec
Work9 response time: 17 sec, turnaround time: 58 sec
Average response time: 9.300000 sec, average turnaround time: 41.600000 sec
Throughput: 0.024038 jobs/sec
CPU utilization: 84.810127%
I/O utilization: 92.405063%
```

- I/O occurence rate: 75%

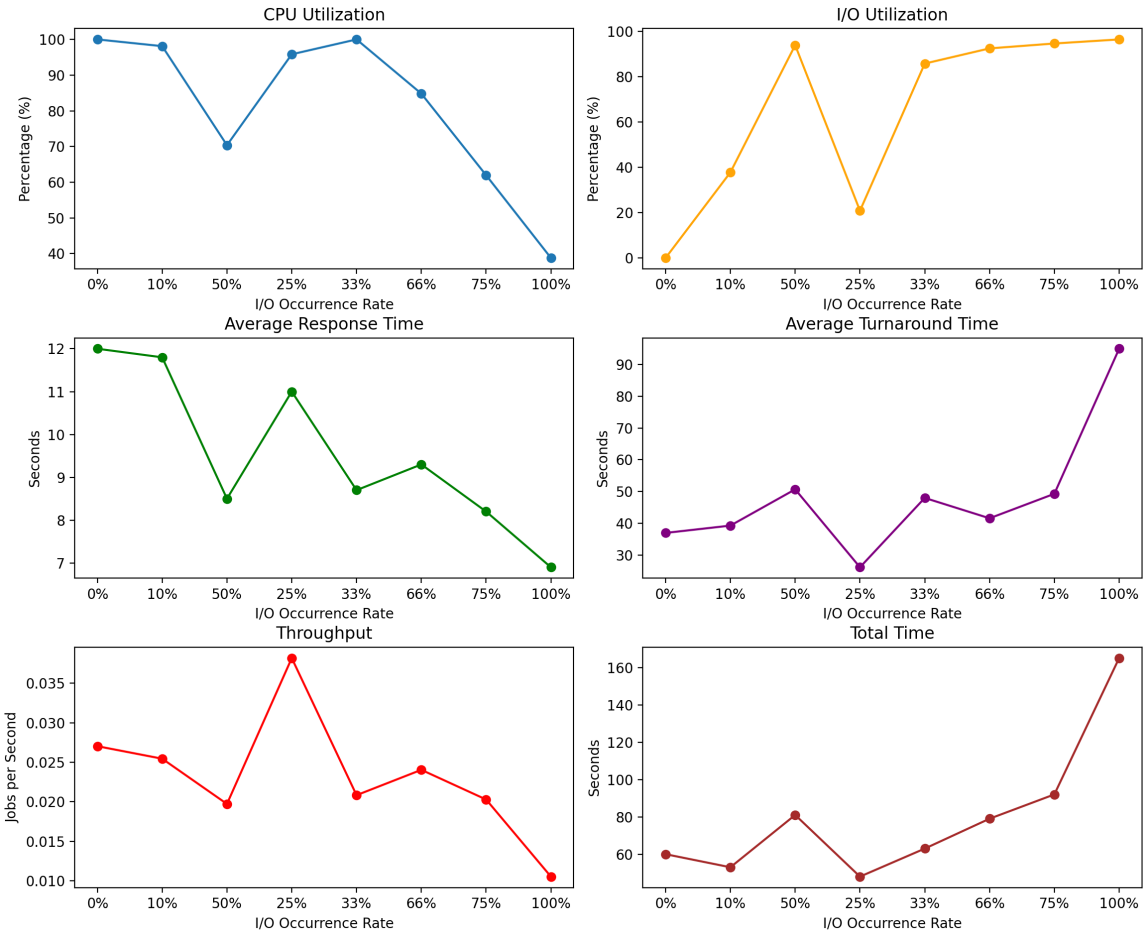
```
[Time: 92 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 28 sec
Work1 response time: 1 sec, turnaround time: 2 sec
Work2 response time: 2 sec, turnaround time: 34 sec
Work3 response time: 5 sec, turnaround time: 53 sec
Work4 response time: 8 sec, turnaround time: 51 sec
Work5 response time: 10 sec, turnaround time: 77 sec
Work6 response time: 11 sec, turnaround time: 37 sec
Work7 response time: 13 sec, turnaround time: 62 sec
Work8 response time: 15 sec, turnaround time: 92 sec
Work9 response time: 17 sec, turnaround time: 57 sec
Average response time: 8.200000 sec, average turnaround time: 49.300000 sec
Throughput: 0.020284 jobs/sec
CPU utilization: 61.956522%
I/O utilization: 94.565217%
```

- I/O occurence rate: 100%

```
[Time: 165 sec] - end of simulation
Work0 response time: 0 sec, turnaround time: 117 sec
Work1 response time: 2 sec, turnaround time: 147 sec
Work2 response time: 3 sec, turnaround time: 5 sec
Work3 response time: 5 sec, turnaround time: 153 sec
Work4 response time: 6 sec, turnaround time: 156 sec
Work5 response time: 7 sec, turnaround time: 38 sec
Work6 response time: 9 sec, turnaround time: 11 sec
Work7 response time: 11 sec, turnaround time: 165 sec
Work8 response time: 12 sec, turnaround time: 74 sec
Work9 response time: 14 sec, turnaround time: 83 sec
Average response time: 6.900000 sec, average turnaround time: 94.900000 sec
Throughput: 0.010537 jobs/sec
```

```
CPU utilization: 38.787879%
I/O utilization: 96.363636%
```

- Result



  - 마찬가지로 각 작업의 수행 시간이 랜덤하게 결정되므로, 일부 표본에서 예상과 다른 결과가 나올 수 있다.
  - 또한 여기서 I/O 발생은 CPU burst가 time quantum 이상일 때 발생하므로, 실제론 해당 확률보다 조금 낮다. (ex. I/O 발생 확률이 100%여도 남은 CPU burst 시간이 time quantum 낮으면 I/O가 발생하지 않음)
  - I/O 발생 확률이 낮은 경우
    - 자명하게 CPU 사용률이 높고, I/O 사용률이 낮다.
    - 평균 응답 시간은 대체로 높고, 평균 반환 시간은 대체로 낮다.
    - 대체로 단위 시간동안 많은 양의 작업을 수행한다. (높은 throughput)
    - 총 소요 시간이 짧은 편이다.
  - I/O 발생 확률이 높은 경우

- 자명하게 CPU 사용률이 낮고, I/O 사용률이 높다.
- 평균 응답 시간은 대체로 낮고, 평균 반환 시간은 대체로 높다.
- 대체로 단위 시간동안 적은 양의 작업을 수행한다. (낮은 throughput)
- 총 소요 시간이 긴 편이다.