|  |  |
| --- | --- |
| 단계 | 완료 여부 |
| * 1. 과제 설명 영상 보기 | O |
| * + 1. List 자료구조 파악: 예제 수행 | O |
| * + 1. 프로세스 정보 읽어오기 | O |
| * + 1. 프로세스 역순회 출력 | O |
| * + 1. 동적할당 해제 | O |
| * 1. 과제 1-1 완료 (JOTA) | O |
| 1-2-0. 과제 설명 영상 보기 | O |
| 1-2-1. idle 프로세스 생성 | O |
| 1-2-2. 도착한 프로세스 확인 및 ready에 넣기 | O |
| 1-2-3. 프로세스 및 코드 동작 종료 확인 | O |
| 1-2-4. while문 종료 조건 확인 | O |
| 1-2-5. 프로세스 및 코드 context switching | O |
| 1-2-6. 다음 loop로 – 변수 증감 | O |
| 1-2-7. ready\_q에 idle 큐 삽입 및 코드 수정 | O |
| * 1. 과제 1-2 완료 (JOTA) | O |
| 1-3-0. 과제 설명 영상 보기 | O |
| 1-3-1. 프로세스 road 및 선택 | O |
| 1-3-2. Context Switching 처리 | O |
| 1-3-3. 프로세스, 코드 및 전체 종료 확인 | O |
| 1-3-4. 코드 실행 동작 | O |
| 1-3-5. IO 처리 작업 | O |
| * 1. 과제 1-3 완료 (JOTA) | O |

운영체제 hw1 보고서  
컴퓨터공학부 202219352 김담은  
  
**# 자율 진도표**

과제 1-2, 과제 1-3 비교 내용 보고서에 포함하기

**# os1-1.c**

[주요 코드 부분 서술]

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

- fread로 int 자료형 3개 만큼 각각 pid, arrival\_time, code\_bytes 변수에 값을 읽어온다.

- operations을 code\_bytes/2만큼 동적할당하고 개수만큼, 각각 값을 읽어온다.

JOTA 1-1 제출 결과 캡쳐

텍스트, 소프트웨어, 컴퓨터 아이콘, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

어려웠던 점 및 해결 방안

- 어려웠던 점 : 처음 process 구조체의 값을 읽어올 때, code\_tuple을 어떻게 읽어올 것인가에 대한 고민이 있었다.

- 해결 방안 : fread의 사용법을 찾아보고, fread(cur, sizeof(int), 3, stdin)

다음과 같이 작성 시 구조체의 처음 부분에 int 자료형만큼 3개 값을 읽어올 수 있다는 것을 알게 되었다. 이후 code\_bytes/2만큼 code를 동적할당 하여 code\_tuple을 읽어올 수 있었다.

**# os1-2.c**

[주요 코드 부분 서술]

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

\* 작업 종료 확인 부분

- CPU 작업, IO 작업이 끝났다면 cpu\_clock, io\_clock을 초기화하고 program\_counter를 증가

- program\_counter가 마지막+1을 가리킨다면 해당 프로세스를 ready 큐에서 빼기

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

\* context switching 처리

- context switching이 이뤄지면 clock 및 idle\_time에 10씩을 더하고, 프로세스 및 작업 코드 변경

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

\* 다음 loop로 진행 처리

- cpu 작업의 경우 cpu\_clock에 1을 더하고, io 작업의 경우 io\_clock 및 idle\_time에 1을 더한다.

- while문 끝 부분에서 clock에 1을 더한다.

[JOTA 1-2 제출 결과 캡쳐]

텍스트, 소프트웨어, 웹 페이지, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

[어려웠던 점 및 해결 방안]

- 어려웠던 점: 코드 동작 확인, Context Switching, 종료 확인 등 각 동작의 순서를 정하는 부분이 어려웠다.

- 해결 방안: clock의 동작을 0부터 1까지 길게 생각해보며 동작의 순서를 고민해보았다. Clock이 동작할 때 프로세스의 처리 과정을 머릿속으로 그려가며 코드를 작성하여 해결했다.

**# os1-3.c**

[주요 코드 부분 서술]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

\* Context Switching

- context switching을 하면 clock에 10을 더하고 wait queue의 프로세스들의 작업을 10만큼 처리해준다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

\* IO 처리

- IO 처리 시 wait\_q를 순회하며 각 프로세스의 length에 -1씩 해주고, length가 0이 됐다면 ready\_q에 넣어주고, wait\_q에서 빼준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

\* 코드 실행 동작

- cpu 작업과 io 작업을 구분해 process 작업 종료를 확인한다.

[JOTA 1-3 제출 결과 캡쳐]

텍스트, 소프트웨어, 웹 페이지, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

텍스트, 소프트웨어, 컴퓨터 아이콘, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

[어려웠던 점 및 해결 방안]

- 어려웠던 점: context switching, 코드 동작 처리, 코드 종료 확인, 프로세스 종료 확인, IO 작업 처리 등 작업 과정의 처리 순서를 정하는 데에 어려움이 있었다.

- 해결 방안: 결과를 보며 코드를 수정해 나갔으나, 여전히 틀리는 부분이 있으며 맞을 때도 있다. 완벽하게 해결하지 못했다.