운영체제 프로젝트 중간보고서

1. 개요 및 설명

a. 개요

프로세스는 컴퓨터에서 실행 중인 프로그램을 의미한다. 프로그램이 디스크에 저장된 정적인 코드라면, 프로세스는 이 프로그램이 실행되어 메모리에 적재되고 CPU에 의해 수행되는 동적인 상태를 나타낸다.

b. 프로세스 구조

- **프로세스 ID (PID)**

  각 프로세스는 고유한 식별자(PID)를 갖는다.

- **메모리 공간**
  - Stack : 함수 호출과 관련된 Local Variable 및 Return Address 저장.
  - Heap : 동적 메모리 할당 공간.
  - Data : Global Variable이 저장되는 공간.
  - Codes : 실행할 명령어들이 저장되는 공간.

- **프로세스 상태**

  프로세스는 생성(Created), 준비(Ready), 실행(Running), 대기(Waiting), 종료(Terminated) 등의 상태를 가진다.

  생성 : 프로세스가 처음 만들어지는 상태.
  준비 : 실행을 기다리는 상태.
  실행 : 현재 CPU에서 실행 중인 상태.
  대기 : 어떤 이벤트를 기다리는 상태. (예: I/O , wait … etc)
  종료 : 실행이 완료되거나 강제로 종료된 상태.

c. 프로젝트 구현 목표

- Process 관리자

- IPC / Shared Memory

- 추가내용 있으면 추가 예정

2. 프로젝트 진행 상황

a. 코드 설명

- 이 코드는 간단한 프로세스 관리 시스템을 구현한 것으로, 사용자가 프로세스를 생성하고 종료할 수 있으며, 현재 실행 중인 프로세스 목록을 확인할 수 있는 기능을 제공한다.

- 헤더 파일 및 변수 정의

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <readline/readline.h>

#define MAX_PROCESSES 10

typedef struct {
    int pid;        // 프로세스 ID
    int *memory;    // 할당된 메모리
} Process;
Process processes[MAX_PROCESSES];  // 프로세스 배열
int num_processes = 0;             // 생성된 프로세스 수
```

- 프로세스 생성 함수

```c
void create_process(int memory_size) {
    if (num_processes < MAX_PROCESSES) {
        // 메모리 할당
        processes[num_processes].memory = (int *)malloc(memory_size * sizeof(int));
        if (processes[num_processes].memory == NULL) {
            printf("메모리 할당에 실패했습니다.\n");
            return;
        }
        processes[num_processes].pid = num_processes + 1; // PID는 1부터 시작
        num_processes++;
        printf("프로세스가 생성되었습니다. PID: %d, 할당된 메모리 크기: %d\n",
processes[num_processes-1].pid, memory_size);
    } else {
        printf("더 이상 프로세스를 생성할 수 없습니다. 최대 프로세스 개수를 초과했습니다.\n");
    }
}
```

memory_size 크기의 메모리를 할당받아 새로운 프로세스를 생성.

프로세스 ID는 1부터 시작하며 순차적으로 증가하며, 메모리 할당에 실패할 경우 오류 메시지를 출력한다.

- 프로세스 종료 함수

```
void terminate_process(int pid) {
    int i, found = 0;
    for (i = 0; i < num_processes; i++) { //find all processs , thread
        if (processes[i].pid == pid) {
            printf("프로세스 PID %d 종료됨. 할당된 메모리 해제됨\n", processes[i].pid);
            // 할당된 메모리 해제
            free(processes[i].memory);
            // 프로세스 제거
            for (; i < num_processes - 1; i++) {
                processes[i] = processes[i + 1];
            }
            num_processes--;
            found = 1;
            break;
        }
    }
```

주어진 pid를 가진 프로세스를 종료하며, 프로세스가 종료되면 할당된 메모리를 해제한다.

입력된 PID가 유효하지 않으면 오류 메시지를 출력한다.

- 프로세스 목록 출력 함수

```
void show_processes() {
    printf("현재 실행 중인 프로세스:\n");
    if (num_processes == 0) {
        printf("실행 중인 프로세스가 없습니다.\n");
        return;
    }
    for (int i = 0; i < num_processes; i++) {
        printf("PID: %d, 할당된 메모리 크기: %p\n", processes[i].pid, (void
*)processes[i].memory);
    }
}
```

- 실행(main)

```
void process_manager() {
  printf("=====================================\n");
  show_processes();
  printf("=====================================\n");
  char *input;
  int memory_size, pid;
  while(1) {
    input = readline("Process Manager : (종료 : exit / 프로세스 생성 : create / 프로세스
종료 : terminate / 프로세스 목록 : list) : ");
    if (strcmp(input,"exit") == 0)
    {
          break;
    }
    else if (strcmp(input, "create") == 0)
    {
    printf("할당할 메모리 크기를 입력하세요: ");
    scanf("%d", &memory_size);
    create_process(memory_size);
    }
    else if (strcmp(input, "terminate") == 0)
    {
    printf("종료할 프로세스의 PID 를 입력하세요: ");
    scanf("%d", &pid);
    terminate_process(pid);
    }
    else if (strcmp(input, "list") == 0)
    {
      show_processes();
    }
    else
    {
    printf("알 수 없는 명령어입니다.\n");
    }
  }
}
```

3. 향후 프로젝트 진행 계획

- IPC / Shared Memory 구현.
- Process State 관련된 Code 추가.