Design Document (Assignment 2)

Process Status

```
typedef enum process_status {
    RUNNING = 0,
        # Add the READY state
    READY = 1,
        # Add the STOPPED state
    STOPPED = 2,
    TERMINATED = 3
} process_status;
```

Process Record

```
typedef struct process_record {
    pid_t pid;
    process_status status;
    int total_runtime; # Total expected runtime of the process
    int remaining_runtime; # Remaining runtime of the process
} process_record;
enum {
        # Set to 64 as per requirements
    MAX_PROCESSES = 64
};
process_record process_records[MAX_PROCESSES];
```

Process Management Functions

```
void scheduler(void) {
        # Iterate through process_records and identify the process with the least
        # remaining run time from all the READY processes and change its status to RUNNING.
}
```

```
void perform_run(char* args[], int runtime) {
    # New processes are added to the READY queue and scheduler() is called
    scheduler();
}
```

```
void perform_stop(char* args[]) {
    # Change status of specified PID to STOPPED and call scheduler()
    scheduler();
}
```

```
void perform_kill(char* args[]) {
    # Search for process with given PID, and send a SIGTERM signal to terminate it. # Set its status to TERMINATED
        # Call scheduler function to decide which function should be running.
    scheduler();
}
```

```
void perform_list(char* args[]) {
    # Loop through all processes and print out PID and status code for each process
}
```

```
void perform_resume(char* args[]) {
    # Search for process with given PID, and changes status from STOPPED to READY
        # Call scheduler() to decide which function should be running.
    scheduler();
}
```

```
void perform_exit(void) {
    printf("bye!\n");
}
```

Main

```
int main(void) {
    # Enters into an infinite while loop, processes user input and calls the appropriate function
    # based on the command entered (run, stop, resume, list, kill, exit). The program terminates
    # when the exit command is entered.
}
```