Slip 18

Program 1: Shell with `list` Command

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>

void list(char *option, char *dirname) {
   DIR *dir;
   struct dirent *entry;
   dir = opendir(dirname);
   if (dir == NULL) {
      printf("Directory %s not found.\n", dirname);
      return;
   }

   if (strcmp(option, "f") == 0) {
      while ((entry = readdir(dir)) != NULL) {
         if (entry->d_type == DT_REG) {
            printf("%s\n", entry->d_name);
         }
      }
   } else if (strcmp(option, "i") == 0) {
      while ((entry = readdir(dir)) != NULL) {
         if (entry->d_type == DT_REG) {
            printf("%s\t%lu\n", entry->d_name, entry->d_fileno);
         }
      }
   }

   closedir(dir);
}

int main() {
   char command[100], *args[10];
   while (1) {
      printf("\nmyshell$ ");
      fgets(command, 100, stdin);
```

```
        command[strlen(command) - 1] = '\0';  // Remove newline

        char *token = strtok(command, " ");
        int i = 0;
        while (token != NULL) {
            args[i++] = token;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;

        if (strcmp(args[0], "list") == 0) {
            list(args[1], args[2]);
        } else if (strcmp(args[0], "exit") == 0) {
            exit(0);
        } else {
            int pid = fork();
            if (pid == 0) {
                execvp(args[0], args);
                exit(0);
            } else {
                wait(NULL);
            }
        }
    }
    return 0;
}
```

Program 2: Round Robin Scheduling Algorithm

```
#include <stdio.h>

struct process {
    int pid;
    int burst_time;
    int remaining_time;
    int turnaround_time;
    int waiting_time;
};

void round_robin(struct process p[], int n, int quantum) {
```

```
    int time = 0, completed = 0;

    while (completed != n) {
        for (int i = 0; i < n; i++) {
            if (p[i].remaining_time > 0) {
                if (p[i].remaining_time > quantum) {
                    time += quantum;
                    p[i].remaining_time -= quantum;
                } else {
                    time += p[i].remaining_time;
                    p[i].turnaround_time = time;
                    p[i].remaining_time = 0;
                    completed++;
                }
            }
        }
    }
}

void calculate_times(struct process p[], int n) {
    int total_turnaround = 0, total_waiting = 0;
    printf("\nPID\tTurnaround Time\tWaiting Time\n");

    for (int i = 0; i < n; i++) {
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
        printf("%d\t%d\t\t%d\n", p[i].pid, p[i].turnaround_time, p[i].waiting_time);
        total_turnaround += p[i].turnaround_time;
        total_waiting += p[i].waiting_time;
    }

    printf("\nAverage Turnaround Time: %.2f\n", (float)total_turnaround / n);
    printf("Average Waiting Time: %.2f\n", (float)total_waiting / n);
}

int main() {
    int n, quantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);
```

```
    struct process p[n];
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter burst time for process %d: ", p[i].pid);
        scanf("%d", &p[i].burst_time);
        p[i].remaining_time = p[i].burst_time;
    }

    printf("Enter time quantum: ");
    scanf("%d", &quantum);

    round_robin(p, n, quantum);
    calculate_times(p, n);

    return 0;
}
```