fork exec wait

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main() {
pid_t pid;
pid = fork();
if (pid < 0) {
perror("Fork failed");
exit(1);
} else if (pid == 0) {
printf("This is the child process.\n");
execlp("/bin/ls", "ls", NULL);
perror("Exec failed");
exit(1);
} else {
printf("This is the parent process. Waiting for the child
to finish...\n");
wait(NULL);
printf("Child process has finished. Parent process
continues.\n");
}
return 0;
}
```

cp:--

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
FILE source, destination;
char ch;
if (argc != 3) {
printf("Usage: %s <source> <destination>\n", argv[0]);
exit(1);
}
source = fopen(argv[1], "rb");
if (source == NULL) {
perror("Error opening source file");
exit(1);
}
destination = fopen(argv[2], "wb");
if (destination == NULL) {
perror("Error opening destination file");
fclose(source);
exit(1);
}
while ((ch = fgetc(source)) != EOF) {
fputc(ch, destination);
}
fclose(source);
fclose(destination);
return 0;}
```

ls:--

```
#include <stdio.h>
#include <dirent.h>
int main() {
struct dirent *entry;
DIR *dp = opendir(".");
if (dp == NULL) {
perror("Error opening directory");
return 1;
}
while ((entry = readdir(dp)) != NULL) {
printf("%s\n", entry->d_name);
}
closedir(dp);
return 0;
}
```

threads:--

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void* print_message(void* arg) {
printf("Thread %d is running\n", (int)arg);
return NULL;
}
int main() {
pthread_t threads[5];
int thread_ids[5];
for (int i = 0; i < 5; i++) {
thread_ids[i] = i + 1;
if (pthread_create(&threads[i], NULL, print_message,
(void*)&thread_ids[i]) != 0) {
printf("Error creating thread\n");
return 1;
}
}
for (int i = 0; i < 5; i++) {
pthread_join(threads[i], NULL);
}
return 0;
}
```

FCFS:----

```
#include <stdio.h>
struct Process {
int id;
int arrival_time;
int burst_time;
int waiting_time;
int turnaround_time;
int completion_time;
};
void calculateFCFS(struct Process p[], int n) {
p[0].waiting_time = 0;
p[0].turnaround_time = p[0].burst_time;
p[0].completion_time = p[0].arrival_time +
p[0].burst_time;
for (int i = 1; i < n; i++) {
p[i].waiting_time = p[i-1].completion_time -
p[i].arrival_time;
if (p[i].waiting_time < 0) p[i].waiting_time = 0;
p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
p[i].completion_time = p[i].arrival_time +
p[i].waiting_time + p[i].burst_time;
}
}
void displayTable(struct Process p[], int n) {
int total_wt = 0, total_tat = 0;
```

```
printf("\nID\tArrival Time\tBurst Time\tWaiting
Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id,
p[i].arrival_time, p[i].burst_time, p[i].waiting_time,
p[i].turnaround_time);
total_wt += p[i].waiting_time;
total_tat += p[i].turnaround_time;
}
printf("Average Waiting Time: %.2f\n", (float)total_wt/n);
printf("Average Turnaround Time: %.2f\n",
(float)total_tat/n);
}
int main() {
int n;
FILE *file;
file = fopen("input.txt", "r");
if (file == NULL) {
printf("Error opening file\n");
return 1;
}
fscanf(file, "%d", &n);
struct Process p[n];
for (int i = 0; i < n; i++) {
p[i].id = i + 1;
fscanf(file, "%d %d", &p[i].arrival_time, &p[i].burst_time);
p[i].waiting_time = -1; // -1 indicates not yet calculated
```

```
}
fclose(file);
printf("\nFCFS Scheduling:\n");
calculateFCFS(p, n);
displayTable(p, n);
return 0;
}
```

SJF:--non Preemptive

```
#include <stdio.h>
struct Process {
int id;
int arrival_time;
int burst_time;
int waiting_time;
int turnaround_time;
int completion_time;
};
void calculateSJF(struct Process p[], int n) {
int completed = 0, time = 0;
int min_burst, idx;
while (completed < n) {
min_burst = 9999;
idx = -1;
for (int i = 0; i < n; i++) {
if (p[i].arrival_time <= time && p[i].waiting_time == -1 &&
p[i].burst_time < min_burst) {
min_burst = p[i].burst_time;
idx = i;
}
}
if (idx != -1) {
p[idx].waiting_time = time - p[idx].arrival_time;
if (p[idx].waiting_time < 0) p[idx].waiting_time = 0;
```

```
p[idx].turnaround_time = p[idx].waiting_time +
p[idx].burst_time;
p[idx].completion_time = time + p[idx].burst_time;
time = p[idx].completion_time;
completed++;
} else {
time++;
}
}
}
void displayTable(struct Process p[], int n) {
int total_wt = 0, total_tat = 0;
printf("\nID\tArrival Time\tBurst Time\tWaiting
Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id,
p[i].arrival_time, p[i].burst_time, p[i].waiting_time,
p[i].turnaround_time);
total_wt += p[i].waiting_time;
total_tat += p[i].turnaround_time;
}
printf("Average Waiting Time: %.2f\n", (float)total_wt/n);
printf("Average Turnaround Time: %.2f\n",
(float)total_tat/n);
}
int main() {
int n;
```

```
FILE *file;
file = fopen("input.txt", "r");
if (file == NULL) {
printf("Error opening file\n");
return 1;
}
fscanf(file, "%d", &n);
struct Process p[n];
for (int i = 0; i < n; i++) {
p[i].id = i + 1;
fscanf(file, "%d %d", &p[i].arrival_time, &p[i].burst_time);
p[i].waiting_time = -1; // -1 indicates not yet calculated
}
fclose(file);
printf("\nSJF Scheduling:\n");
calculateSJF(p, n);
displayTable(p, n);
return 0;
}
```

SJF PREEMPTIVE...:-.

```
#include <stdio.h>

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
    int completion_time;
};

void calculateSJF(struct Process p[], int n) {
    int completed = 0, time = 0, idx;
    int min_remaining_time;
    while (completed < n) {
        min_remaining_time = 9999;
        idx = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival_time <= time && p[i].remaining_time
> 0 && p[i].remaining_time < min_remaining_time) {
                min_remaining_time = p[i].remaining_time;
                idx = i;
            }
        }
```

```
        if (idx != -1) {
            p[idx].remaining_time--;
            if (p[idx].remaining_time == 0) {
                p[idx].completion_time = time + 1;
                p[idx].turnaround_time =
p[idx].completion_time - p[idx].arrival_time;
                p[idx].waiting_time = p[idx].turnaround_time -
p[idx].burst_time;
                completed++;
            }
        }
        time++;
    }
}

void displayTable(struct Process p[], int n) {
    int total_wt = 0, total_tat = 0;
    printf("\nID\tArrival Time\tBurst Time\tWaiting
Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id,
p[i].arrival_time, p[i].burst_time, p[i].waiting_time,
p[i].turnaround_time);
        total_wt += p[i].waiting_time;
        total_tat += p[i].turnaround_time;
    }
    printf("Average Waiting Time: %.2f\n", (float)total_wt/
```

```
n);
    printf("Average Turnaround Time: %.2f\n",
(float)total_tat/n);
}

int main() {
    int n;
    FILE *file;
    file = fopen("input.txt", "r");
    if (file == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    fscanf(file, "%d", &n);
    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        fscanf(file, "%d %d", &p[i].arrival_time,
&p[i].burst_time);
        p[i].remaining_time = p[i].burst_time;
        p[i].waiting_time = 0;
        p[i].turnaround_time = 0;
        p[i].completion_time = 0;
    }
    fclose(file);
    printf("\nPreemptive SJF Scheduling:\n");
    calculateSJF(p, n);
```

```
    displayTable(p, n);
    return 0;
}
```

READER AND WRITER SDEMAPHORES:--

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define MAX_READERS 10
#define MAX_WRITERS 10
sem_t mutex, rw_mutex;
int read_count = 0;
int write_count = 0;
void reader(void arg) {
int reader_id = ((int )arg);
sem_wait(&mutex);
read_count++;
if (read_count == 1)
sem_wait(&rw_mutex);
sem_post(&mutex);
printf("Reader %d is reading.\n", reader_id);
sleep(1);
sem_wait(&mutex);
read_count--;
if (read_count == 0)
sem_post(&rw_mutex);
sem_post(&mutex);
printf("Reader %d is done reading.\n", reader_id);
return NULL;
}
```

```
void writer(void arg) {
int writer_id = ((int )arg);
sem_wait(&rw_mutex);
write_count++;
printf("Writer %d is writing.\n", writer_id);
sleep(2);
write_count--;
sem_post(&rw_mutex);
printf("Writer %d is done writing.\n", writer_id);
return NULL;
}
int main() {
int num_readers, num_writers;
FILE *file = fopen("input.txt", "r");
if (file == NULL) {
printf("Error opening file\n");
return 1;
}
fscanf(file, "%d", &num_readers);
fscanf(file, "%d", &num_writers);
fclose(file);
sem_init(&mutex, 0, 1);
sem_init(&rw_mutex, 0, 1);
pthread_t readers[num_readers], writers[num_writers];
int reader_ids[num_readers], writer_ids[num_writers];
for (int i = 0; i < num_readers; i++) {
reader_ids[i] = i + 1;
```

```
pthread_create(&readers[i], NULL, reader, &reader_ids[i]);
}
for (int i = 0; i < num_writers; i++) {
writer_ids[i] = i + 1;
pthread_create(&writers[i], NULL, writer, &writer_ids[i]);
}
for (int i = 0; i < num_readers; i++) {
pthread_join(readers[i], NULL);
}
for (int i = 0; i < num_writers; i++) {
pthread_join(writers[i], NULL);
}
sem_destroy(&mutex);
sem_destroy(&rw_mutex);
return 0;
}
```