// Eric Goulart da Cunha - 2110878
// João Pedro Biscaia Fernandes - 2110361

Devem ser entregues os códigos fonte (interpretador, escalonador e programas de teste) e um relatório (em .pdf) indicando que programas serão executados em seu teste, instruções para compilação, a ordem de entrada para o escalonador e a ordem de execução determinada pelo escalonador (estas são informações da saída dos programas), juntamente com uma análise crítica sobre o que, de fato, ocorreu (se a ordem de execução dos programas foi a esperada, incluindo o que funciona e o que não funciona no seu trabalho). Essas explicações também serão objeto de avaliação.

Entrada esperada:
O formato da entrada precisa ser, necessariamente, desse formato, no exec.txt:

```
1   Run p1
2   Run p2
3   Run p3 I=10 D=5
4   Run p4 I=12 D=5
5   Run p5 I=15 D=20
6   Run p6 P=1
7   Run p7 P=7
```

Para os programas de execução, compile o arquivo loop.c (loop infinito)

Informações de execução
Para rodar o programa, compile o escalonador (escalonador.c) e o interpretador(interpretador.c), e execute o interpretador. Caso necessário, após executar o interpretador é possível, em outro terminal, executar o escalonador(somente se ja tiver rodado o interpretador).

Explicação do código:

Escalonador:

Alocação de memoria compartilhada:

```c
// aloca a memória compartilhada
segmento = shmget(80, sizeof(Processo)*MAX_PROCESSOS, IPC_CREAT| 0666);

if (segmento == -1) {
  fprintf(stderr, "Não foi possível alocar memória\n");
  exit(1);
}
// associa a memória compartilhada ao processo
vetor_programas = (Processo *)shmat(segmento, 0, 0); // comparar o retorno com -1
```

Criação de processos filhos:

```c
while( i < MAX_PROCESSOS && (strcmp(vetor_programas[i].nome,"")!=0)) {
  if ((processo_exec[i].pid = fork()) < 0) {
    fprintf(stderr, "Erro ao criar filho");
    exit(1);
  }
  if (processo_exec[i].pid == 0) {
    /*inicializa a execução do programa filho, se puder ser executado*/
    char comando[15] = "./";
    char *args[] = {strcat(comando,vetor_programas[i].nome), NULL}; // Argumentos do
rograma
    execvp(args[0], args); // Execução do programa
    exit(EXIT_SUCCESS);
  } else { /*Processo pai*/
    kill(processo_exec[i].pid, SIGSTOP);
  }
  i++;
}
```

Loop de execução:

```c
float sec, print_sec;
while (TRUE) {
  gettimeofday(&current_time, NULL);
  sec = ((current_time.tv_sec - start_time.tv_sec));
  print_sec = (current_time.tv_sec - start_time.tv_sec) % 60;
  if(ready->status == RUNNING){
    if(realtime_finished(ready, print_sec) == TRUE){
      ready->status = COMPLETED;
    }else if(ready->process.prioridade > -1){
      if(sec - ready->started_time >=5){
        kill(ready->pid, SIGSTOP);
        ready->status = COMPLETED;
      }
    }else if(ready->process.prioridade == -1 && ready->process.inicio == -1){
      if(sec - ready->started_time >= 1){
        ready->status = COMPLETED;
        kill(ready->pid, SIGSTOP);
      }
    }
    if(realtime_quer_comecar(fila_realtime, print_sec) == TRUE && ready-
rocess.inicio == -1){
      ready = coloca_realtime_comeco(ready, print_sec, fila_realtime);
      fila_realtime->status = RUNNING;
      kill(ready->pid, SIGCONT);
      ready->status = RUNNING;
    }
  }
```

```
    if(ready->status == COMPLETED){
      if(ready->process.prioridade > -1){
        ready = pop_fila(ready);
      }else{
        if(ready->process.inicio == -1){
          ready = push_fila(ready->process,ready, ready->pid, WAITING);
        }else if(ready->process.inicio > -1){
          ready->status = WAITING;
          fila_realtime->status = WAITING;
          fila_realtime = adiciona_realtime_denovo(fila_realtime);
        }
        ready = pop_fila(ready);
      }
      if(realtime_quer_comecar(fila_realtime, print_sec) == TRUE && ready-
>process.inicio == -1){
        ready = coloca_realtime_comeco(ready, print_sec, fila_realtime);
        fila_realtime->status = RUNNING;
        kill(ready->pid, SIGCONT);
        ready->status = RUNNING;
      }

    }
    if(ready->status == WAITING){
      ready = trata_waiting(ready, print_sec, sec, fila_realtime);
    }
    if(ready->status == PAUSED){
      kill(ready->pid, SIGCONT);
      ready->status = RUNNING;
    }
    printf("%.1f %s\n",print_sec + 1,ready->process.nome);

    sleep(1);
```

Como pode ser visto, usamos filas para rodar cada programa em loop, se não for prioridade. Se for prioridade, após 5 segundos rodando, é retirado da fila e não roda mais.

Em cada elemento da fila, ele pode estar, rodando, esperando, pausado ou completado.
Se está rodando, é verificado se o tempo limite dele acabou(round robin é 1 segundo, realtime é tempo de duração e prioridade é 5 segundos), e coloca no status de completado.
Se completado, verifica se é round robin. Se sim, é adicionado na fila principal(ready), se é realtime, é removido da fila e guardado para o próximo minuto. Se está pausado, retorna a ser executado.
No final de cada loop, ao remover um item do topo da fila, é verificado se o próximo item da lista está esperando. Se sim, o inicia e atualiza seu status para rodando.

Desafios:
Nossos maiores desafios ao decorrer do trabalho foram o uso de memória compartilhada e a lógica do nosso código, que precisou ser refeita. Porém conseguimos superar esses desafios.

Saída do terminal abaixo. Acreditamos que agora o código rode como deveria.

Interpretador:

```
Run p1

Run p2

Run p3 I=10 D=5

Run p4 I=12 D=5

Run p5 I=15 D=20

Run p6 P=1

Run p7 P=7
```

Escalonador:

```
--------------------Escalonador--------------------

1.0 p1
2.0 p2
3.0 p1
4.0 p2
5.0 p1
6.0 p2
7.0 p6
8.0 p6
9.0 p6
10.0 p3
11.0 p3
12.0 p3
13.0 p3
14.0 p3
15.0 p5
16.0 p5
17.0 p5
18.0 p5
19.0 p5
20.0 p5
21.0 p5
22.0 p5
23.0 p5
24.0 p5
25.0 p5
26.0 p5
27.0 p5
28.0 p5
29.0 p5
30.0 p5
31.0 p5
32.0 p5
33.0 p5
34.0 p5
35.0 p6
36.0 p7
37.0 p7
38.0 p7
39.0 p7
40.0 p7
```

```
40.0 p7
41.0 p1
42.0 p2
43.0 p1
44.0 p2
45.0 p1
46.0 p2
47.0 p1
48.0 p2
49.0 p1
50.0 p2
51.0 p1
52.0 p2
53.0 p1
54.0 p2
55.0 p1
56.0 p2
57.0 p1
58.0 p2
59.0 p1
60.0 p2
1.0 p1
2.0 p2
3.0 p1
4.0 p2
5.0 p1
6.0 p2
7.0 p1
8.0 p2
9.0 p1
10.0 p3
11.0 p3
12.0 p3
13.0 p3
14.0 p3
```