Trabalho 1

Érica Oliveira Regnier - 2211893
Hanna Epelboim Assunção - 2310289

Versão com I/O:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <time.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/stat.h>

#define NUM_PROCESSOS 3
#define TIMESLICE 1 // Timeslice de 1 segundo


// Estrutura do no
typedef struct No {
   int processo;
   struct No* next;
} No;

// Estrutura da fila
typedef struct Fila {
   No* cabeca;
   No* fim;
} Fila;

int cpu = 0;
int processos[NUM_PROCESSOS]; // PIDs dos processos
int processo_atual = NUM_PROCESSOS-1; // Índice do processo que está
executando
int processo_block;
```

```c
// int espera[NUM_PROCESSOS] = {0}; //se processo block --> 1
int terminados[NUM_PROCESSOS] = { 0 }; //se processo terminado --> 1
int term = 0;
int flag = 0;
int* PC;
int* espera;
Fila* block;
////////////////////////////////////////////////////////////////////////
//////////////////////////////
//FUNCOES DE FILA
No* criaNo(int processo) {
   No* no = (No*)malloc(sizeof(No));
   if (!no) {
       perror("Erro ao alocar memoria para o no");
       exit(EXIT_FAILURE);
   }
   no->processo = processo;
   no->next = NULL;
   return no;
}

int vazia(Fila* queue) {
   return queue->cabeca == NULL;
}

Fila* criaFila() {
   Fila* fila = (Fila*)malloc(sizeof(Fila));
   if (!fila) {
       perror("Erro ao alocar memoria para a fila");
       exit(EXIT_FAILURE);
   }
   fila->cabeca = NULL;
   fila->fim = NULL;
   return fila;
}

void insereFila(Fila* fila, int processo) {
   No* no = criaNo(processo);
```

```
    if (vazia(fila)) {
        fila->cabeca = no;
        fila->fim = no;
    }
    else {
        fila->fim->next = no;
        fila->fim = no;
    }
}

No* removeFila(Fila* fila) {
    if (vazia(fila)) {
        //printf("Fila esta vazia\n");
        return NULL;
    }
    No* temp = fila->cabeca;
    fila->cabeca = fila->cabeca->next;
    if (fila->cabeca == NULL) {
        fila->fim = NULL;
    }
    return temp;
}

No* veInicio(Fila* fila) {
    if (vazia(fila)) {
        printf("Fila esta vazia\n");
        return NULL;
    }
    return fila->cabeca;
}


void imprimeFila(Fila* fila) {
    if (vazia(fila)) {
        printf("Fila está vazia\n");
        return;
    }
    No* atual = fila->cabeca;
```

```
    printf("Fila de processos espera: ");
    while (atual != NULL) {
        //printf("%d ", atual->processo);
        atual = atual->next;
    }
    printf("\n");
}


void liberaFila(Fila* fila) {
    while (!vazia(fila)) {
        No* temp = removeFila(fila);
        free(temp);
    }
    free(fila);
}
////////////////////////////////////////////////////////////////////////
////////////////////////////////////


void irq0_handler(int sig) {


    printf("\n\n\nTempo CPU: %d\n", cpu);
    cpu++;
   if(cpu!=1 && !espera[processo_atual] && !terminados[processo_atual]
){
        kill(processos[processo_atual], SIGSTOP); // Pausar o processo
atual
        printf("Parando processo %d\n", processo_atual);
    }

    // Avançar para o próximo processo
    processo_atual = (processo_atual + 1) % NUM_PROCESSOS;

    for (int i = 0; i < NUM_PROCESSOS; i++) {
        // printf("oi - %d\n", espera[i]);
        if (espera[i]) {
```

```
            espera[i]--;
            //printf("%d --> %d", i, espera[i]);
            if (!espera[i]) {
                //printf("livreee\n");
                kill(getpid(), SIGHUP);
            }
        }
    }


    for (int i = 0; i < NUM_PROCESSOS; i++) {

        if (!espera[processo_atual] && !terminados[processo_atual]) {
            //printf("\nespera proc atua%d\n",
espera[processo_atual]);

            printf("Processo %d, PC=%d\n", processo_atual,
PC[processo_atual]);//de fora

            kill(processos[processo_atual], SIGCONT);   // Continuar o
próximo processo
            //  printf("p atual%d\n",processo_atual);
            //printf("pre:%d  ",PC[processo_atual]);
            PC[processo_atual]--;
            //printf("pos:%d/n",PC[processo_atual]);
            break;
        }

        if(term == NUM_PROCESSOS){
            printf("Todos processos terminaram!");
        }

    }
}


void syscall_handler(int signo) {
    //printf("\n%d\n", id);
```

```
   kill(processos[processo_atual],SIGSTOP);
   printf("syscall: Entrada IO\n");
   insereFila(block, processo_atual);
   //imprimeFila(block);
   espera[processo_atual] = 3;
   printf("Parando processo %d (por IO)\n", processo_atual);

}


void irq1_handler(int sig) {
   printf("IRQ1: Saida IO\n");
   No* desbloqueado = removeFila(block);
   if (desbloqueado != NULL) {
   int proc_id = desbloqueado->processo;
   printf("Processo %d voltou da operação de IO\n", proc_id);
   kill(processos[proc_id], SIGCONT); // Retoma o processo da fila de
bloqueados
   free(desbloqueado);
}

   //kill(processos[processo_atual], SIGCONT);

   //removeFila(block);
}



void processo_funcao(int id) {
   while (PC[id] > 0) {


       sleep(1); // Simula a execução
       if ((PC[processo_atual]) == 5) {
           //printf("%d  %d", id, getpid());
           //syscall_handler(id, getpid());
           kill(getppid(), SIGUSR2);
       }
```

```
    }
    terminados[id] = 1;
    term++;
    exit(0); // Termina o processo
}




void intercontroller_sim() {
    while (1) {
        sleep(TIMESLICE); // Emula IRQ0 a cada 1 segundo
        kill(getppid(), SIGUSR1); // Envia sinal para o KernelSim
    }
}


int main() {
    signal(SIGUSR1, irq0_handler); // Registrar o tratador de IRQ0
    signal(SIGHUP, irq1_handler); // Registrar fim do tratador de
tempo de io
    signal(SIGUSR2, syscall_handler); // Registrar inicio do tratador
de tempo de io
    block = criaFila();
    int segmento = shmget(IPC_PRIVATE, (sizeof(int) * NUM_PROCESSOS),
IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    if (segmento == -1) {
        perror("Erro ao alocar memória compartilhada");
        exit(1);
    }

    int segmentoEspera = shmget(IPC_PRIVATE, (sizeof(int) *
NUM_PROCESSOS), IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    if (segmentoEspera == -1) {
        perror("Erro ao alocar memória compartilhada");
        exit(1);
    }
```

```
   // associa a memória compartilhada ao processo
   PC = (int*)shmat(segmento, NULL, 0); // comparar o retorno com -1
   if (PC == (void*)-1) {
       perror("Erro ao associar a memória compartilhada");
       exit(1);
   }

   for (int i = 0; i < NUM_PROCESSOS; i++) {
       PC[i] = 7;
   }

   espera = (int*)shmat(segmentoEspera, NULL, 0); // comparar o
retorno com -1
   if (espera == (void*)-1) {
       perror("Erro ao associar a memória compartilhada");
       exit(1);
   }


   for (int i = 0; i < NUM_PROCESSOS; i++) {
       espera[i] = 0;
   }

   // Criar processos de aplicação
   for (int i = 0; i < NUM_PROCESSOS; i++) {
       if ((processos[i] = fork()) == 0) {
           // Código do processo de aplicação
           processo_funcao(i);

       }
   }



   // Pausar todos os processos no início
   for (int i = 0; i < NUM_PROCESSOS; i++) {
       kill(processos[i], SIGSTOP);
   }
```

```
    // Criar o controlador de interrupções (InterControllerSim)
    int pidInter = fork();
    if (pidInter == 0) {
        intercontroller_sim();
        exit(0);
    }


    // Inicializar o primeiro processo
    kill(processos[processo_atual], SIGCONT);


    // KernelSim espera a finalização de todos os processos
    for (int i = 0; i < NUM_PROCESSOS; i++) {
        wait(NULL);
    }


    kill(pidInter, SIGKILL);
    liberaFila(block);
    shmdt(PC);
    shmdt(espera);

    shmctl(segmento, IPC_RMID, NULL);
    shmctl(segmentoEspera, IPC_RMID, NULL);

 return 0;
}
```

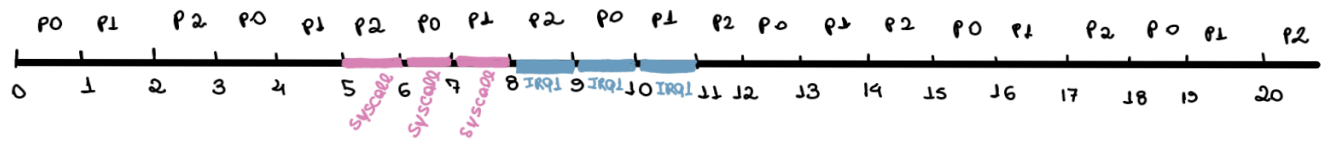
Saída:

```
[c2310289@torres ~/Downloads]$ gcc -Wall -o t123 erica+hanna.c
[c2310289@torres ~/Downloads]$ ./t123



Tempo CPU: 0
Processo 0, PC=7



Tempo CPU: 1
Parando processo 0
Processo 1, PC=7



Tempo CPU: 2
Parando processo 1
Processo 2, PC=7



Tempo CPU: 3
Parando processo 2
Processo 0, PC=6



Tempo CPU: 4
Parando processo 0
Processo 1, PC=6



Tempo CPU: 5
Parando processo 1
Processo 2, PC=6
syscall: Entrada IO
Parando processo 2 (por IO)



Tempo CPU: 6
Processo 0, PC=5
syscall: Entrada IO
Parando processo 0 (por IO)



Tempo CPU: 7
Processo 1, PC=5
syscall: Entrada IO
Parando processo 1 (por IO)
```

```
Tempo CPU: 8
IRQ1: Saida IO
Processo 2 voltou da operação de IO
Processo 2, PC=5



Tempo CPU: 9
Parando processo 2
IRQ1: Saida IO
Processo 0 voltou da operação de IO
Processo 0, PC=4



Tempo CPU: 10
Parando processo 0
IRQ1: Saida IO
Processo 1 voltou da operação de IO
Processo 1, PC=4



Tempo CPU: 11
Parando processo 1
Processo 2, PC=4



Tempo CPU: 12
Parando processo 2
Processo 0, PC=3



Tempo CPU: 13
Parando processo 0
Processo 1, PC=3



Tempo CPU: 14
Parando processo 1
Processo 2, PC=3



Tempo CPU: 15
Parando processo 2
Processo 0, PC=2
```

```
Tempo CPU: 16
Parando processo 0
Processo 1, PC=2


Tempo CPU: 17
Parando processo 1
Processo 2, PC=2


Tempo CPU: 18
Parando processo 2
Processo 0, PC=1


Tempo CPU: 19
Parando processo 0
Processo 1, PC=1


Tempo CPU: 20
Parando processo 1
Processo 2, PC=1
[c2310289@torres ~/Downloads]$
```

Linha do tempo (saída esperada):



Explicação:

O objetivo do código é simular um escalonador de processos que utiliza o mecanismo de time-sharing (timeslice de 1 segundo) com suporte a bloqueios de entrada/saída (I/O). O código implementa processos filhos que executam instruções um máximo de vezes. Durante a execução, processos podem ser bloqueados para realizar I/O e são retomados ao fim dessa operação. As filas são implementadas para armazenar processos que estão bloqueados aguardando operações de I/O. O código utiliza estruturas de dados encadeadas para gerenciar a fila de bloqueados.

São criados processos filhos - nesse caso 3. Cada um simula um programa que diminui seu contador de programa (PC) até atingir zero, indicando o término da execução. Cada processo pode ser bloqueado se precisar de I/O. O código usa sinais para simular interrupções (IRQs) e chamadas de sistema (syscalls). Utilizamos SIGUSR1, SIGUSR2, SIGHUP, SIGCONT e SIGSTOP

As funções como criaNo(), insereFila(), removeFila() e imprimeFila() são usadas para manipular a fila de processos bloqueados que aguardam o término de I/O. A fila armazena nós que contêm o identificador do processo. Quando o processo finaliza sua operação de I/O, ele é removido da fila e pode voltar à execução.

O irq0_handler simula a interrupção do relógio (timer interrupt), responsável por trocar o processo ativo a cada timeslice de 1 segundo. O processo atual é pausado com SIGSTOP e o próximo processo da fila de prontos é retomado com SIGCONT. O syscall_handler simula uma chamada de sistema de entrada/saída (I/O). Quando um processo atinge um ponto de execução onde precisa de I/O, paramos ele com SIGSTOP e o colocamos na fila de bloqueados. O irq1_handler simula o término de uma operação de I/O. Remove o processo da fila de bloqueados e o retoma para continuar sua execução.

A função processo_funcao(int id) define a execução de cada processo. Cada processo inicia com um PC (contador de programa) igual a 7(definido na main). O código utiliza memória compartilhada para armazenar o contador de programa (PC) e o estado de espera dos processos (tempo de espera para desbloqueio de I/O). A memória compartilhada é alocada com shmget e acessada via shmat. A função intercontroller_sim() emula um controlador de interrupções que dispara um sinal (SIGUSR1) a cada 1 segundo, simulando a interrupção do relógio que controla a troca de contexto entre os processos.

Fluxo de Execução:

Primeiro são criados três processos filhos e cada um inicia sua execução. Os processos são inicialmente pausados até que o primeiro timeslice ocorra. A cada 1 segundo, o manipulador de IRQ (irq0_handler) troca o processo em execução, pausando o processo atual e retomando o próximo na fila circular de processos. Se um processo chega a um ponto onde precisa de I/O, ele é interrompido, movido para a fila de bloqueados, e um contador de espera é iniciado. Após

um período de tempo (simulado como 3 segundos), o processo é desbloqueado e pode ser retomado pelo escalonador. Quando todos os processos terminam sua execução (isto é, quando todos os PC chegam a 0), o simulador termina.

Timeslice:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <time.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/stat.h>



#define NUM_PROCESSOS 3
#define TIMESLICE 1 // Timeslice de 1 segundo



int cpu = 0;
int processos[NUM_PROCESSOS]; // PIDs dos processos
int processo_atual = NUM_PROCESSOS-1; // Índice do processo que está
executando
// int espera[NUM_PROCESSOS] = {0}; //se processo block --> 1
int terminados[NUM_PROCESSOS] = { 0 }; //se processo terminado --> 1
int term = 0;
int* PC;
```

```
void irq0_handler(int sig) {


    printf("\n\n\nTempo CPU: %d\n", cpu);
    cpu++;
   if(cpu!=1 && !terminados[processo_atual] ){
        kill(processos[processo_atual], SIGSTOP); // Pausar o
processo atual
        printf("Parando processo %d\n", processo_atual);
    }


    // Avançar para o próximo processo
    processo_atual = (processo_atual + 1) % NUM_PROCESSOS;

    for (int i = 0; i < NUM_PROCESSOS; i++) {


        if (!terminados[processo_atual]) {
            //printf("\nespera proc atua%d\n",
espera[processo_atual]);


            printf("Processo %d, PC=%d\n", processo_atual,
PC[processo_atual]);//de fora


            kill(processos[processo_atual], SIGCONT);   // Continuar
o próximo processo
  //          printf("p atual%d\n",processo_atual);



//            printf("pre:%d  ",PC[processo_atual]);

            PC[processo_atual]--;
//            printf("pos:%d/n",PC[processo_atual]);
```

```
            break;
        }

        if(term == NUM_PROCESSOS){
            printf("Todos processos terminaram!");
        }

    }
}


void processo_funcao(int id) {
    while (PC[id] > 0) {



        sleep(1); // Simula a execução

    }
    terminados[id] = 1;
    term++;
    exit(0); // Termina o processo
}

void intercontroller_sim() {
    while (1) {
        sleep(TIMESLICE); // Emula IRQ0 a cada 1 segundo
        kill(getppid(), SIGUSR1); // Envia sinal para o KernelSim
    }
}


int main() {
    signal(SIGUSR1, irq0_handler); // Registrar o tratador de IRQ0

    int segmento = shmget(IPC_PRIVATE, (sizeof(int) * NUM_PROCESSOS),
IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    if (segmento == -1) {
```

```
        perror("Erro ao alocar memória compartilhada");
        exit(1);
    }



    // associa a memória compartilhada ao processo
    PC = (int*)shmat(segmento, NULL, 0); // comparar o retorno com -1
    if (PC == (void*)-1) {
        perror("Erro ao associar a memória compartilhada");
        exit(1);
    }



    for (int i = 0; i < NUM_PROCESSOS; i++) {
        PC[i] = 7;
    }




    // Criar processos de aplicação
    for (int i = 0; i < NUM_PROCESSOS; i++) {
        if ((processos[i] = fork()) == 0) {
            // Código do processo de aplicação
            processo_funcao(i);

        }
    }



    // Pausar todos os processos no início
    for (int i = 0; i < NUM_PROCESSOS; i++) {
        kill(processos[i], SIGSTOP);
    }
    // Criar o controlador de interrupções (InterControllerSim)
    int pidInter = fork();
```

```
    if (pidInter == 0) {
        intercontroller_sim();
        exit(0);
    }




    // Inicializar o primeiro processo
    kill(processos[processo_atual], SIGCONT);



    // KernelSim espera a finalização de todos os processos
    for (int i = 0; i < NUM_PROCESSOS; i++) {
        wait(NULL);
    }

    kill(pidInter, SIGKILL);


shmdt(PC);
    shmctl(segmento, IPC_RMID, NULL);



  return 0;
}
```

Comandos:

```
[c2310289@acara t1]$ gcc -Wall -o tsl VersaoTimeSlice.c
[c2310289@acara t1]$ ./tsl
```

Saída 1:

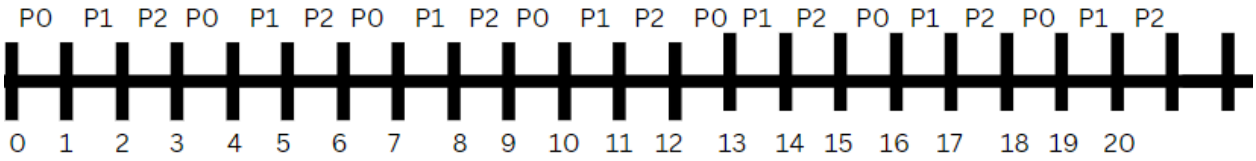
```
Tempo CPU: 0
Processo 0, PC=7


Tempo CPU: 1
Parando processo 0
Processo 1, PC=7


Tempo CPU: 2
Parando processo 1
Processo 2, PC=7


Tempo CPU: 3
Parando processo 2
Processo 0, PC=6


Tempo CPU: 4
Parando processo 0
Processo 1, PC=6


Tempo CPU: 5
Parando processo 1
Processo 2, PC=6
```

```
Tempo CPU: 6
Parando processo 2
Processo 0, PC=5


Tempo CPU: 7
Parando processo 0
Processo 1, PC=5


Tempo CPU: 8
Parando processo 1
Processo 2, PC=5


Tempo CPU: 9
Parando processo 2
Processo 0, PC=4


Tempo CPU: 10
Parando processo 0
Processo 1, PC=4


Tempo CPU: 11
Parando processo 1
Processo 2, PC=4
```

```
Tempo CPU: 12
Parando processo 2
Processo 0, PC=3


Tempo CPU: 13
Parando processo 0
Processo 1, PC=3


Tempo CPU: 14
Parando processo 1
Processo 2, PC=3


Tempo CPU: 15
Parando processo 2
Processo 0, PC=2


Tempo CPU: 16
Parando processo 0
Processo 1, PC=2


Tempo CPU: 17
Parando processo 1
Processo 2, PC=2
```

```
Tempo CPU: 18
Parando processo 2
Processo 0, PC=1


Tempo CPU: 19
Parando processo 0
Processo 1, PC=1


Tempo CPU: 20
Parando processo 1
Processo 2, PC=1


...Program finished with exit code 0
Press ENTER to exit console.
```

Saída 1 esperada:



EXPLICAÇÃO:

O código implementa uma simulação básica de um sistema operacional que gerencia a execução de três processos usando uma estratégia de escalonamento round-robin com um timeslice de 1 segundo. Cada processo é executado até esgotar o número de iterações, e o sistema vai alternando entre os processos, pausando e retomando cada um de acordo com o tempo de CPU que é definido.

A função irq0_handler emula uma interrupção de timer para alternar entre processos. A função processo_função simula um processo que roda um máximo de vezes (nesse caso 7 vezes). A variável PC salva o contexto do processo e torna possível que o processo funcione esse máximo de vezes. O intercontroller_sim simula o controlador de interrupções que gera sinais periódicos para alternar entre os processos.

O código funciona exatamente como previsto pela nossa linha do tempo, como visto na “Saída 1 esperada”.