Relatório do Trabalho 1 de INF 1036 - Sistemas Operacionais

Primeiro Trabalho - Interpretador e Escalonador

Data: 29/10/2023

Alunos:
Leo Lomardo - matrícula 2020201
Lucas Lucena - matrícula 2010796

Objetivo:

Criar um escalonador que opere processos tanto com política Real-Time quanto com política Round-Robin. As instruções dos processos são passadas por um arquivo .txt. Existe um programa interpretador que lê essas instruções e se comunica com um programa escalonador, passando por meio de mensagens as instruções de cada um dos processos a ser executado.

Estrutura do programa:

Para rodar o trabalho, primeiro é necessário compilar todos os arquivos (interpretador.c, escalonador.c, prog1.c, prog2.c, prog3.c, prog4.c e prog5.c). Depois, você executa o escalonador, que ficará esperando por mensagens enviadas para ele. Então, você executa o interpretador, em um outro terminal, que vai analisar o arquivo exec.txt, lendo as instruções de cada processo, enviando cada uma delas por mensagem para o escalonador. Com isso, o escalonador começará a operar.

Interpretador:

- Abre o arquivo "exec.txt" e conta o número total de comandos no arquivo. Isso é feito pela função contaComandos.
- Cria uma fila de mensagens para comunicação com o escalonador, usando a função msgget.
- Lê o arquivo linha por linha, extrai informações sobre os comandos, cria estruturas Comando correspondentes e envia essas estruturas para o escalonador através da fila de mensagens. Envia a mensagem linha por linha, não todas de uma vez.

Escalonador:

- Inicializa uma fila de comandos e dois semáforos, fila_mutex e exec_mutex, para proteger o acesso concorrente à fila e à execução de processos.
- Define um tratador de sinal para o alarme, que encerrará o escalonador após um determinado tempo.
- No loop principal, aguarda novos comandos vindos do produtor através da fila de mensagens e os adiciona à fila de comandos. Ele também verifica se há conflitos de escalonamento para comandos RealTime.
- O escalonador verifica periodicamente a fila de comandos e decide se deve iniciar a execução de comandos com base no tipo (RealTime ou RoundRobin). Ele garante que, no máximo, um comando RealTime seja executado ao mesmo tempo.
- Quando a execução de um comando RealTime é concluída, ele passa para o próximo comando na fila. Quando um comando RoundRobin é executado, ele é interrompido após um segundo e o próximo comando é executado.
- O escalonador é encerrado após um determinado tempo (que no caso é de 120 segundos, como pedido no enunciado), de acordo com o alarme definido.

Processos 1-5:

- Processos infinitos muito simples que imprimem o valor total que foi usado enquanto processo, além de qual o nome daquele processo.
- Com isso, podemos ver claramente qual o processo está sendo executado em qual momento, e por qual vai ser substituído.

Este código é uma implementação simples de um sistema de escalonamento de processos que lida com dois tipos de comandos: RealTime e RoundRobin. Ele demonstra a comunicação entre processos usando filas de mensagens e o uso de semáforos para sincronização..

Solução:

Código das estruturas utilizadas (estruturas.h):

```c
#include <stdio.h>
#include <stdlib.h>

#define REAL_TIME 1
#define ROUND_ROBIN 0
```

```c
typedef struct comando{
char nome_programa[46];
int momento_inicio;
int tempo_duracao;
int tipo; // 0 para RoundRobin, 1 para RealTime-
int index;
int pid;
} Comando;


typedef struct processoatual{
int escalonado;
Comando proc;
}ProcessoAtual;

typedef struct mensagem{
long tipo;
Comando comando;
} Mensagem;
```

Código do programa “interpretador.c”:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#include "estruturas.h"

int main() {
key_t chave = 7000;
int fila_mensagens;
int num_comandos = 0;
```

```
char line[100];

// Conta o número total de comandos no arquivo
FILE *arquivo1 = fopen("exec.txt", "r");
if (arquivo1 == NULL) {
printf("Não foi possível abrir o arquivo.\n");
return 1;
}
num_comandos = contaComandos(arquivo1);
Comando comandos[num_comandos];
printf("Número total de comandos: %d\n", num_comandos);
fclose(arquivo1);

FILE *arquivo = fopen("exec.txt", "r");
if (arquivo == NULL) {
printf("Não foi possível abrir o arquivo.\n");
return 1;
}

// Cria uma fila de mensagens para comunicação com o escalonador
fila_mensagens = msgget(chave, IPC_CREAT | 0666);
if (fila_mensagens == -1) {
perror("Erro ao criar a fila de mensagens");
exit(1);
}

int comando_idx = 0;

while (fgets(line, sizeof(line), arquivo)) {
char comando[8];

if (sscanf(line, "Run %s", comando) == 1) {
Comando novo_comando;
strcpy(novo_comando.nome_programa, comando);
if (strstr(line, "I=") && strstr(line, "D=")) {
// É um comando RealTime
int momento_inicio, tempo_duracao;
sscanf(line, "Run %*s I=%d D=%d", &momento_inicio, &tempo_duracao);
novo_comando.momento_inicio = momento_inicio;
novo_comando.tempo_duracao = tempo_duracao;
```

```
novo_comando.tipo = REAL_TIME; // RealTime
novo_comando.index = comando_idx;
} else {
// É um comando RoundRobin
novo_comando.momento_inicio = 0; // Valor padrão para RoundRobin
novo_comando.tempo_duracao = 0; // Valor padrão para RoundRobin
novo_comando.tipo = ROUND_ROBIN; // RoundRobin
novo_comando.index = comando_idx;
}

comandos[comando_idx] = novo_comando;
comando_idx++;

// Envia o comando para o escalonador através da fila de mensagens
if (msgsnd(fila_mensagens, &novo_comando, sizeof(Comando), 0) == -1) {
perror("Erro ao enviar mensagem para o escalonador");
exit(1);
}
}
}

fclose(arquivo); // Fecha o arquivo quando terminar de usá-lo

return 0;
}

int contaComandos(FILE *arquivo){
char line[35];
int num_comandos = 0;

while (fgets(line, sizeof(line), arquivo) != NULL) {
if (sscanf(line, "Run %*s") == 0) {
num_comandos++;
}
}
return num_comandos;
}
```

Código do programa “escalonador.c”:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <sys/wait.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/time.h>
#include "estruturas.h"
#include <semaphore.h>

#define MAX_COMANDOS 100
int rodando = 1;

typedef struct {
Comando comandos[MAX_COMANDOS];
int tamanho;
} FilaComandos;

void trataAlarme(int signum){
printf("Tempo de execucao do escalonador esgotado\n");
rodando = 0;
}

sem_t fila_mutex; // Semáforo para proteger a fila
sem_t exec_mutex; // Semáforo para controlar a execução dos processos

void inicializarFila(FilaComandos *fila) {
fila->tamanho = 0;
sem_init(&fila_mutex, 0, 1); // Inicializa o semáforo com um recurso
sem_init(&exec_mutex, 0, 1); // Inicializa o semáforo de execução com um
recurso
}
```

```c
void pararComando(Comando* comando) {
signal(comando->pid, SIGSTOP);
}

int conflitoEscalonamento(const Comando* novo, const FilaComandos* fila) {
for (int i = 0; i < fila->tamanho; i++) {
const Comando* existente = &fila->comandos[i];
if (existente->tipo == REAL_TIME && novo->tipo == REAL_TIME) {
if (novo->momento_inicio >= existente->momento_inicio &&
novo->momento_inicio < (existente->momento_inicio +
existente->tempo_duracao)) {
return 1; // Conflito de tempo de início
}
}
}
return 0;
}

void adicionarComando(FilaComandos *fila, Comando comando) {
if(comando.tipo == REAL_TIME &&
comando.momento_inicio+comando.tempo_duracao > 60){
printf("Comando RealTime ultrapassaria do minuto e não foi adicionado à
fila.\n");
return;
}
if (conflitoEscalonamento(&comando, fila)) {
printf("Conflito de escalonamento para o comando %s\n",
comando.nome_programa);
return;
}
printf("Adicionando comando %s à fila de comandos\n",
comando.nome_programa);
comando.pid = fork();
if (comando.pid == 0) {
// Processo filho
execve(comando.nome_programa, NULL, NULL);
perror("Erro ao executar o comando");
sleep(1);
signal(comando.pid, SIGSTOP);
exit(1);
```

```
} else if (comando.pid > 0) {
// Processo pai
sem_wait(&fila_mutex); // Bloqueia o semáforo para atualizar a fila
fila->comandos[fila->tamanho] = comando;
fila->tamanho++;
sem_post(&fila_mutex); // Libera o semáforo após a atualização
} else {
perror("Erro ao criar o processo filho");
}
}

int main() {
key_t chave = 7000;
int fila_mensagens;
FilaComandos fila;
Comando* comando_atual;

// Inicializa a fila de comandos
inicializarFila(&fila);

// Variáveis para obter o tempo atual
time_t tempo_atual;
struct tm* info_tempo;
int segundos_atuais;

// Comunicando com o escalonador
fila_mensagens = msgget(chave, 0666);
if (fila_mensagens == -1) {
perror("Erro ao acessar a fila de mensagens");
exit(1);
}

printf("Escalonador iniciado\n");

signal (SIGALRM, trataAlarme);
alarm (120);


while (rodando) {
sleep(1);
```

```
Comando mensagem;
if (msgrcv(fila_mensagens, &mensagem, sizeof(mensagem), 0, IPC_NOWAIT) !=
-1) {
// Recebeu uma mensagem
printf("Recebeu um comando\n");
adicionarComando(&fila, mensagem);
}

time(&tempo_atual);
info_tempo = localtime(&tempo_atual);
segundos_atuais = info_tempo->tm_sec;

int executando_real_time = 0;

for (int i = 0; i < fila.tamanho; i++) {
time(&tempo_atual);
info_tempo = localtime(&tempo_atual);
segundos_atuais = info_tempo->tm_sec;
if (fila.comandos[i].tipo == REAL_TIME) {
time(&tempo_atual);
segundos_atuais = info_tempo->tm_sec;
int diferenca_tempo = fila.comandos[i].momento_inicio - segundos_atuais;
if (diferenca_tempo <= 0 && diferenca_tempo >= -1) {
sem_wait(&exec_mutex); // Bloqueia o semáforo de execução
printf("Segundo de início: %d - segundo atual: %d \n",
fila.comandos[i].momento_inicio, segundos_atuais);
printf("Diferença de tempo: %d\n", abs(diferenca_tempo));
// É hora de executar o comando RealTime
executando_real_time = 1;
signal(fila.comandos[i].pid, SIGCONT);
sleep(fila.comandos[i].tempo_duracao);
signal(fila.comandos[i].pid, SIGSTOP);
sem_post(&exec_mutex); // Libera o semáforo de execução
}
} else if (fila.comandos[i].tipo == ROUND_ROBIN && executando_real_time ==
0) {
sem_wait(&exec_mutex); // Bloqueia o semáforo de execução
// Rodando os comandos RoundRobin
signal(fila.comandos[i].pid, SIGCONT);
sleep(1);
```

```
signal(fila.comandos[i].pid, SIGSTOP);
sem_post(&exec_mutex); // Libera o semáforo de execução
}
sleep(1);
for(int j = 0; j < fila.tamanho; j++){
pararComando(&fila.comandos[j]);
}
executando_real_time = 0;
}

}

for (int i = 0; i < fila.tamanho; i++) {
kill(fila.comandos[i].pid, SIGKILL);
}

return 0;
}
```

Código do “prog1.c”:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/time.h>

int main() {
pid_t pid = getpid();
printf("Programa 1 - pid: %d\n", pid);
int tempo_execucao = 0;

while (1) {
printf("Tempo em execução do processo 3: %d segundos\n", tempo_execucao);
sleep(1);
tempo_execucao++;
}
```

```
return 0;
}
```

Código do “prog2.c”:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/time.h>

int main() {
pid_t pid = getpid();
printf("Programa 2 - pid: %d\n", pid);
int tempo_execucao = 0;

while (1) {
printf("Tempo em execução do processo 2: %d segundos\n", tempo_execucao);
sleep(1);
tempo_execucao++;
}

return 0;
}
```

Código do “prog3.c”:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/time.h>

int main() {
pid_t pid = getpid();
printf("Programa 3 - pid: %d\n", pid);
int tempo_execucao = 0;
```

```
while (1) {
printf("Tempo em execução do processo 3: %d segundos\n", tempo_execucao);
sleep(1);
tempo_execucao++;
}

return 0;
}
```

Código do “prog4.c”:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/time.h>

int main() {
pid_t pid = getpid();
printf("Programa 4 - pid: %d\n", pid);
int tempo_execucao = 0;

while (1) {
printf("Tempo em execução do processo 4: %d segundos\n", tempo_execucao);
sleep(1);
tempo_execucao++;
}

return 0;
}
```

Código do “prog5.c”:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/time.h>
```

```c
int main() {
pid_t pid = getpid();
printf("Programa 5 - pid: %d\n", pid);
int tempo_execucao = 0;

while (1) {
printf("Tempo em execução do processo 5: %d segundos\n", tempo_execucao);
sleep(1);
tempo_execucao++;
}

return 0;
}
```

E o texto do “exec.txt” que usamos foi:

Run prog1 I=10 D=10
Run prog2 I=30 D=10
Run prog3 I=35 D=5
Run prog4
Run prog5

Para compilar e executar os programas, fizemos o seguinte:
gcc -Wall -o prog1 prog1.c
gcc -Wall -o prog2 prog2.c
gcc -Wall -o prog3 prog3.c
gcc -Wall -o prog4 prog4.c
gcc -Wall -o prog5 prog5.c
gcc -Wall -o interpretador interpretador.c
gcc -Wall -o escalonador escalonador.c
./escalonador
(Escalonador começa a rodar em um terminal)
Abrir outro terminal
./interpretador (no novo terminal)

Um exemplo de saída que tivemos, para a visualização dos processo ativos, foi:

```
Escalonador iniciado
Recebeu um comando
Adicionando comando prog1 à fila de comandos
Executando processo 1
Tempo em execução do processo 1: 0 segundos
Tempo em execução do processo 1: 1 segundos
Recebeu um comando
Adicionando comando prog2 à fila de comandos
Programa 2 - pid: 3454
Tempo em execução do processo 2: 0 segundos
Tempo em execução do processo 1: 2 segundos
Tempo em execução do processo 2: 1 segundos
Tempo em execução do processo 1: 3 segundos
Tempo em execução do processo 2: 2 segundos
Tempo em execução do processo 1: 4 segundos
Recebeu um comando
Conflito de escalonamento para o comando prog3
Tempo em execução do processo 2: 3 segundos
Tempo em execução do processo 1: 5 segundos
Tempo em execução do processo 2: 4 segundos
Tempo em execução do processo 1: 6 segundos
Tempo em execução do processo 2: 5 segundos
Tempo em execução do processo 1: 7 segundos
Recebeu um comando
Adicionando comando prog4 à fila de comandos
Programa 4 - pid: 3467
Tempo em execução do processo 4: 0 segundos
Tempo em execução do processo 2: 6 segundos
Tempo em execução do processo 1: 8 segundos
Tempo em execução do processo 4: 1 segundos
Tempo em execução do processo 2: 7 segundos
Tempo em execução do processo 1: 9 segundos
Tempo em execução do processo 4: 2 segundos
Tempo em execução do processo 2: 8 segundos
```

```
Tempo em execução do processo 1: 10 segundos
Tempo em execução do processo 4: 3 segundos
Tempo em execução do processo 2: 9 segundos
Tempo em execução do processo 1: 11 segundos
Tempo em execução do processo 4: 4 segundos
Tempo em execução do processo 2: 10 segundos
Tempo em execução do processo 1: 12 segundos
Recebeu um comando
Adicionando comando prog5 à fila de comandos
Tempo em execução do processo 4: 5 segundos
Tempo em execução do processo 2: 11 segundos
Programa 5 - pid: 3480
Tempo em execução do processo 5: 0 segundos
Tempo em execução do processo 1: 13 segundos
Tempo em execução do processo 4: 6 segundos
Tempo em execução do processo 2: 12 segundos
Tempo em execução do processo 5: 1 segundos
Tempo em execução do processo 1: 14 segundos
Tempo em execução do processo 4: 7 segundos
Tempo em execução do processo 2: 13 segundos
Tempo em execução do processo 5: 2 segundos
Tempo em execução do processo 1: 15 segundos
Tempo em execução do processo 4: 8 segundos
Tempo em execução do processo 2: 14 segundos
Tempo em execução do processo 5: 3 segundos
Tempo em execução do processo 1: 16 segundos
Tempo em execução do processo 4: 9 segundos
Tempo em execução do processo 2: 15 segundos
Tempo em execução do processo 5: 4 segundos
Tempo em execução do processo 1: 17 segundos
Tempo em execução do processo 4: 10 segundos
Tempo em execução do processo 2: 16 segundos
Tempo em execução do processo 5: 5 segundos
Tempo em execução do processo 1: 18 segundos
Tempo em execução do processo 4: 11 segundos
Tempo em execução do processo 2: 17 segundos
```

```
Tempo em execução do processo 5: 6 segundos
Tempo em execução do processo 1: 19 segundos
Tempo em execução do processo 4: 12 segundos
Tempo em execução do processo 2: 18 segundos
Tempo em execução do processo 5: 7 segundos
Tempo em execução do processo 1: 20 segundos
Tempo em execução do processo 4: 13 segundos
Tempo em execução do processo 2: 19 segundos
Tempo em execução do processo 5: 8 segundos
Tempo em execução do processo 1: 21 segundos
Tempo em execução do processo 4: 14 segundos
Tempo em execução do processo 2: 20 segundos
Tempo em execução do processo 5: 9 segundos
. . .

. . .

Tempo em execução do processo 1: 117 segundos
Tempo em execução do processo 2: 116 segundos
Tempo em execução do processo 4: 110 segundos
Tempo em execução do processo 5: 105 segundos
Tempo em execução do processo 1: 118 segundos
Tempo de execução do escalonador esgotado
Tempo em execução do processo 2: 117 segundos
Tempo em execução do processo 4: 111 segundos
Tempo em execução do processo 5: 106 segundos
Tempo em execução do processo 1: 119 segundos
Tempo em execução do processo 4: 112 segundos
Tempo em execução do processo 2: 118 segundos
Tempo em execução do processo 5: 107 segundos
Tempo em execução do processo 1: 120 segundos
Tempo em execução do processo 4: 113 segundos
Tempo em execução do processo 2: 119 segundos
Tempo em execução do processo 5: 108 segundos
Tempo em execução do processo 1: 121 segundos
Tempo em execução do processo 4: 114 segundos
```

```
Tempo em execução do processo 2: 120 segundos
Tempo em execução do processo 5: 109 segundos
Tempo em execução do processo 1: 122 segundos
```

Os comandos são lidos pelo interpretador na ordem correta. Também são enviados do interpretador para o escalonador na ordem correta, com os atributos corretos. O escalonador recebe as instruções dos processos corretamente, e assim que recebe inicia o processo, atribuindo um pid à ele. Porém, a ordem de execução parece que os processos apenas se alternam, como se fossem todos de igual prioridade e Round Robin. Não sabemos o porquê disso acontecer. Os parâmetros dos processos Real Time também não estão sendo obedecidos, e acreditamos que talvez seja por um erro de concorrência.

Observações e conclusões:

O código do interpretador e dos processos funciona perfeitamente. A comunicação entre o interpretador e o escalonador também está funcionando, e o escalonador recebe as mensagens com as instruções dos processos corretamente. Todas as restrições dos processos também estão sendo corretamente analisadas, como se houver um processo que acarretaria execução simultânea, ele não será processado, ou caso o tempo de execução ultrapasse o minuto. Porém estamos com um problema no escalonador no qual às vezes ele começa a executar um processo Real-Time no momento errado, e ultrapassa a duração correta dele. Não temos a menor ideia de porque isso ocorre, tentamos debugar de diversas maneiras diferentes, todas insuscetíveis.