PUC-RIO

Data: 25/10/2017

Professor: Seibel

Alunos: Guilherme de Azevedo Pereira Marques (1220676)

Wellington Bezerra da Silva (1413383)

**Escalonamento em Múltiplos Níveis com Feedback**

**Introdução**

Esse trabalho apresentará a construção de um um interpretador de comandos e um escalonador de programas. O interpretador lê o nome dos programas executáveis com os seus respectivos parâmetros e solicita ao escalonador a execução concorrente dos processos de cada um desses programas. O escalonador executa de forma alternada os processos de acordo com a política de escalonamento que é baseada em níveis de prioridade das filas. Quando um processo for CPU-Bound, ele deverá ter sua prioridade reduzida e quando for I/O-Bound, ele deverá ter sua prioridade aumentada.

**Desenvolvimento**

**1.0 Arquivos enviados:**

A pasta do trabalho enviado contém os seguintes arquivos:

* **main.c:** interpretador + escalonador + TAD de Fila
* **programa.c:** programa a ser executado
  1. **Compilação dos arquivos:**

A compilação dos arquivos "main.c" e "programa.c" foi feita em um Macbook Pro com macOSSierra utilizando o GCC na versão 4.2.1[1]

* 1. **Como compilar cada arquivo:**

Para compilar o arquivo "main.c", foi utilizado o seguinte commando no terminal:

**gcc –Wall –o main main.c**

O arquivo "programa.c" foi compilado de acordo com a necessidade da quantidade de programas a serem enviados para o interpretador. Por exemplo, se forem enviados para o interpretador um "programa0" juntamente com sua rajadas e um "programa1", será necessário criar dois executáveis do arquivo "programa.c":

* **gcc –Wall –o programa0 programa.c**
* **gcc –Wall –o programa1 programa.c**

**2.0 Conhecendo o arquivo "main.c"**

Esse arquivo possui:

* Uma estrutura de um processo(InfoProcesso)
  + Nome do programa
  + Pid do processo
  + Vetor de rajadas – Máximo de 15 rajadas
  + Index do vetor de rajadas
  + Total de rajadas
  + Fila atual
* Uma estrutura de um fila(Fila)
  + Número de elementos da fila
  + Indice para incio da fila
  + Vetor de processos do tipo InfoProcesso
  + Valor de prioridade da fila
  + Time slice da fila

* Funções de fila:
  + **fila\_cria():** função recebe um número de prioridade e um valor de time slice e retorna uma fila se ela for criada com sucesso. Caso contrário, irá retornar um valor igual a NULL.
  + **fila\_vazia():** Recebe um ponteiro para um fila e retorna a quantidade de elementos dentro dela
  + **fila\_insere():** Recebe um ponteiro para uma fila e para um processo e insere esse ultimo no primeiro. Se o tamanho máximo da fila for atingido, o programa será abortado.
  + **fila\_retira():** recebe um ponteiro para uma fila e retorna um ponteiro para o processo do inicio da fila
  + **fila\_libera():** recebe um ponteiro para uma fila e libera todos os elementos do vetor de processo e também o ponteiro da fila
* Função main():
  + Cria as filas f1, f2, f3 e fProcessosIO
  + Instala a rotina de atendimento dos sinais SIGUSR1, SIGUSR2 e SIGCHLD
  + Cria um canal de comunicação entre o interpretador e o escalonador utilizando PIPE
  + Faz um fork() para criar um processo filho identico ao pai
    - **Processo filho: é o Interpretador**
      * Chama a função interpretador()
        + Lê do terminal o nome do programa e as rajadas e coloca essas informações dentro de um vetor de Processos
      * Fecha-se o canal de leitura da pipe(fd[0]) e escreve os dados do vetor de Processos no canal de saída
    - **Processo pai: é o Escalonador**
      * Chama a função escalonador()
      * Fecha-se o canal de escrita da pipe(fd[1]) e lê-se os dados do do vetor de Processos que estão no canal de entrada.
* Função escalonador():
  + Recebe o vetor de processos
  + As informações de cada elemento do vetor processo são passadas para a função criaPrograma() que irá criar um processo
    - Essa função irá fazer um fork(), onde o processo pai irá parar o processo filho no início da execução mandando um sinal SIGSTOP
  + Cada processo, depois de criado, será colocado na fila de maior prioridade.
  + Após isso, o início do escalonamento será iniciado
    - O processo será retirado da fila de maior prioridade e será executado pela **função executando()**
    - Enquanto houver processo em alguma das filas ele será escalonado
* Função Executando()
  + Quando um processo for executado, será necessário verificar se o time slice da fila acabou, se uma das rajadas do processo acabou ou se todas as rajadas já acabaram
    - Se o time slice da fila acabou e ainda tem rajada, esse processo deverá ser enviado para a fila que tem uma prioridade menor do que a que ele estava sendo executado. Se o processo estava sendo executado na fila com menor prioridade e ainda existe rajada, na proxima vez que ele for executado, irá continuar nessa fila.
    - Se a rajada acabou, o processo deverá ser enviado para a fila de processos em I/O, enviando um sinal SIGUSR1 para o escalonador, e deverá ficar por lá durante 3 segundos. Quando esse tempo se passar, um sinal SIGUSR2 deverá ser enviado também para o escalonador avisando que o tempo que o processo ficou na fila de I/O chegou ao fim e o processo deverá ser colocado em uma fila com prioridade superior a que ele estava sendo executado. Se o processo estava sendo executado na fila com maior prioridade e ainda existe rajada, quando ele voltar, deverá continuar na fila que estava.
  + Quando um processo chegar ao seu fim, um sinal SIGCHLD deverá ser enviado ao escalonador
  + Enquanto um processo fica em I/O, outros processos vão sendo executados

**3.0 Algumas especificações:**

**3.1 Rajada negativa**

Se alguma rajada for negativa, ela será ignorada até achar alguma que seja maior que zero. Caso não encontre, o processo não entrará em nenhuma fila.

**3.2 Número de rajadas**

É permitido colocar um número de rajadas entre 1 e 15.

**3.3 Adicionar um novo processo**

Não é permitido adicionar um novo processo enquanto outros estão sendo executados.

**4.0 Problemas que ocorreram**

**3.1 Pipe**

A pipe não estava recebendo um ponteiro e por conta disso as informações não eram passadas para o escalonador. Para solucionar esse problema, foi criado um vetor de ponteiros de InfoProcesso.

**3.2 Arquivo.c**

O programa não estava devolvendo o pid do pai e, por conta disso, o sinal SIGUSR1 não estava sendo enviado. Para solucionar esse problema, a função getpid() foi trocada pela getppid().

**3.3 SIGSTOP**

Foi descoberto que o SIGSTOP devolve um sinal do tipo SIGCHLD e, por conta disso, tivemos que tratar esse sinal para evitar possíveis problemas. Por exemplo, assim que um processo é criado, um sinal sigstop é enviado e, caso ele envie um sigchld, o escalonador irá entender que o processo foi encerrado. Por isso, foi necessário fazer esse tratamento de sinais.

**3.4 Execução**

Foi observado que os resultados são aleatórios e as vezes o programa funciona e as vezes não. As vezes, o programa termina antes, a fila fica vazia sendo que ainda existe um element, entre outros problemas.

**3.5 Sincronia**

Foi observado no trabalho que enquanto um processo saia de I/O, as filas eram verificadas para saber se existia algum element dentro delas. Enquanto essa verificação era feita para cada fila, um processo saia de I/O e era jogado para um fila e ele acabava não sendo retirado de uma fila de maior prioridade que as demais.

**5.0 Testes realizados**

**5.1: 1 programa com 4 rajadas(5, 1, 1,3)**

ROTINAS DE ATENDIMENTO AOS SINAIS USR1 e USR2 OK

ROTINAS DE ATENDIMENTO AOS SINAL SIGCHLD OK

Entre com um número de 1 a 15 de programas:

1

Entre com o número positivo de rajadas que todos os programas terão:

4

Digite o nome do programa 1:

programa0

Digite o tempo da 1 rajada do programa programa0

5

Digite o tempo da 2 rajada do programa programa0

1

Digite o tempo da 3 rajada do programa programa0

1

Digite o tempo da 4 rajada do programa programa0

3

fila 0 criada com sucesso

fila 1 criada com sucesso

fila 2 criada com sucesso

FILA 1

PID: 12093

PID: 12093

FILA 2

PID: 12093

PID: 12093

FILA 3

PID: 12093

PID: 12093

FILA 2

PID: 12093

FILA 1

Fila vazia.

~/Desktop/LALALA 32s

❯ PID: 12093

PID: 12093

PID: 12093

**5.2: 2 programas com 2 rajadas(2,2)**

ROTINAS DE ATENDIMENTO AOS SINAIS USR1 e USR2 OK

ROTINAS DE ATENDIMENTO AOS SINAL SIGCHLD OK

Entre com um número de 1 a 15 de programas:

2

Entre com o número positivo de rajadas que todos os programas terão:

2

Digite o nome do programa 1:

programa0

Digite o tempo da 1 rajada do programa programa0

2

Digite o tempo da 2 rajada do programa programa0

2

Digite o nome do programa 2:

programa1

Digite o tempo da 1 rajada do programa programa1

2

Digite o tempo da 2 rajada do programa programa1

2

fila 0 criada com sucesso

fila 1 criada com sucesso

fila 2 criada com sucesso

FILA 0

PID: 49162

FILA 0

PID: 49163

FILA 1

PID: 49162

FILA 1

PID: 49163

PID: 49162

FILA 0

PID: 49163

FILA 0

PID: 49163

FILA 1

PID: 49162

Programa terminou

NÃO HÁ MAIS PROCESSOS PARA SEREM ESCALONADOS

**5.3: 1 programa com 1 rajada (0)**

ROTINAS DE ATENDIMENTO AOS SINAIS USR1 e USR2 OK

ROTINAS DE ATENDIMENTO AOS SINAL SIGCHLD OK

Entre com um número de 1 a 15 de programas:

1

Entre com o número positivo de rajadas que todos os programas terão:

1

Digite o nome do programa 1:

programa0

Digite o tempo da 1 rajada do programa programa0

0

fila 0 criada com sucesso

fila 1 criada com sucesso

fila 2 criada com sucesso

FILA 0

Programa terminou

Fila vazia.

**5.3: 1 programa com 2 rajadas (2, -1)**

ROTINAS DE ATENDIMENTO AOS SINAIS USR1 e USR2 OK

ROTINAS DE ATENDIMENTO AOS SINAL SIGCHLD OK

Entre com um número de 1 a 15 de programas:

1

Entre com o número positivo de rajadas que todos os programas terão:

2

Digite o nome do programa 1:

programa0

Digite o tempo da 1 rajada do programa programa0

2

Digite o tempo da 2 rajada do programa programa0

-1

fila 0 criada com sucesso

fila 1 criada com sucesso

fila 2 criada com sucesso

FILA 0

PID: 49237

PID: 49237

FILA 1

NÃO HÁ MAIS PROCESSOS PARA SEREM ESCALONADOS

**Conclusão**

A partir desse projeto, os conceitos aprendidos em sala como processo, pipe, sinais, entre outros, puderam ser observados e melhor compreendidos. Começamos a implementar uma memória compartilhada nesse trabalho mas como estava dando muitos problemas decidimos mudar para pipe e percebemos que a implementação foi muito mais fácil. Além disso, foi descoberto que o SIGSTOP manda um sinal SIGCHLD e para resolver esse problema foi necessário tratar esse dois sinais para diferenciá-los.

**Referência Bibliográfica**

[1] Para ver a versão do GCC em um Macbook, basta abrir o terminal e digitar o seguinte commando:

**gcc -dumpversion | cut -f1,2,3 -d.**