Gerente de execução postergada de processos

Trabalho prático da disciplina de Sistemas Operacionais pela Universidade de Brasília (02/2015)

Alunos da professora Alba Melo:
Eduardo Furtado – 09/0111575
Leandro Ramalho – 10/0033571

Ambiente de desenvolvimento, vulgo computadores do LINF:
gcc --version
gcc (SUSE Linux) 4.8.3 20140627 [gcc-4_8-branch revision 212064]
Copyright (C) 2013 Free Software Foundation, Inc.
---
g++ --version
g++ (SUSE Linux) 4.8.3 20140627 [gcc-4_8-branch revision 212064]
Copyright (C) 2013 Free Software Foundation, Inc.

Exemplo de compilação em uma linha:
g++ -Wall remove_postergado.c -o remove_postergado;g++ -Wall shutdown_postergado.c -o shutdown_postergado;g++ -Wall lista_postergados.c -o lista_postergados;g++ -Wall executa_postergado.c -o executa_postergado ;gcc print.c -o print; g++ -Wall servidor.c -o servidor;

Especificação da Implementação:

O projeto está composto por 5 programas:
> executa_postergado
> servidor
> remove_postergado
> lista_postergados
> shutdown_postergado

O servidor, que tem código mais inteligente dos programas, contém um algoritmo Round Robin para gerenciar os processos que recebe através do programa executa_postergado. Cada processo corre de maneira não simultânea por até 10 segundos, que é o tempo de um quantum deste gerenciador. Há uma fila de execução, que foi implementada através de uma lista, onde os elementos são colocados no final e lidos a partir do princípio.

Para executar uma tarefa, é usado execl, por exemplo:
execl(aux->executavel,aux->executavel,EComercial, (char*)0)

Um processo é clonado com a primitiva fork, e o filho começa já parado através de um SIGSTOP (no fim da lista de execução), para garantir que não ira executar junto do pai.

Quando um processo termina sua execução, ele passa a não participar dessa lista e vai para uma nova, com informações determinadas por outra estrutura de dados, com as estatísticas de sua execução.

Além disso, o servidor também tem um código em seu loop principal que busca mensagens na fila e da a elas o tratamento segundo as informações contidas nelas.

O programa executa postergado também está responsável por dizer ao servidor quantas vezes uma tarefa será executada e de quanto em quanto tempo.

É possível ver os processos que aguardam inicio de sua execução com o programa lista_postergados.

Além de ver os processos, é possível também os remover, utilizando o programa

remove_postergado, que remove os processos referentes a um mesmo *job*.

O programa shutdown_postergado desativa o servidor, cancela e lista os jobs que ainda não foram executados e ao final imprime estatísticas sobre os processos que foram executados.

Sobre as estruturas de dados utilizadas:

As estruturas de dados utilizadas estãono arquivo dados.h.

Para a conversa usando mensagens, existe a estrutura:

```
typedef struct{
	long int mtype;
	long int jobId;
	char executavel[300];
	unsigned int vezes;
	unsigned int hora;
	unsigned int min;
} t_msg;
```

mtype está especificado no próprio dados.h, e pode ser:

(1) TIPOAGENDAMENTO – usado pelo programa executa_postergado
(2) TIPOCANCELAMENTO – usado pelo programa remove_postergado
(3) TIPOKILLSERVER – usado pelo programa shutdown_postergado
(4) TIPOLISTARPROCESSOS – usado pelo programa _postergados

A estrutura que contém informações sobre um processo é:

```
typedef struct t_processo{
	long int jobId;
	char executavel[300];
	unsigned int vezes;
	unsigned int deltaHora;
	unsigned int deltaMin;
	unsigned int horaInsercao;
	unsigned int minInsercao;
	unsigned int minstamp;
	unsigned int startHora;
	unsigned int startMinuto;
	std::vector <int> pid; // pid dos processos deste job
	struct t_processo* prox;
} t_processo;
```

A seguinte estrutura guarda dados sobre os processos já executados, para mostrar durante shutdown_postergado:

```
typedef struct t_estatisticaProcesso{
	long int jobId;
	int pid;
	char executavel[300];
	unsigned int horaInicio;
	unsigned int minInicio;
	unsigned int horaStart;
	unsigned int minStart;
	unsigned int horaFinal;
	unsigned int minFinal;
	struct t_estatisticaProcesso* prox;
```

} t_estatisticaProcesso;

Sobre os mecanismos de comunicação interprocessos utilizados:

Os se comunicam com o servidor através de uma fila de mensagens, que está organizada utilizando a scruct apresentada acima. O servidor utiliza a função msgrcv:

```
while(msgrcv(key_msg,&msgrecebida,size_msg,0,IPC_NOWAIT) > 0) { ... }
```

Enquanto que os programas usam msgsnd:

```
msgsnd(key_msg,&msg1,size_msg, tipo);
```