UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

Implementação de um μnúcleo - Trabalho prático 2 - ENTREGA : 03/05/2012

1. Objetivo

O objetivo deste trabalho é o desenvolvimento de um μnúcleo de um sistema operacional. Este sistema operacional deverá oferecer capacidades para manipulação de processos (criação, execução, trocas de contexto e término) e uma primitiva de sincronização (*join*).

2. Comportamento do μnúcleo

O μnúcleo deverá ser capaz de suportar simultâneamente no máximo 128 processos. O diagrama de transição de processos deverá possui os seguintes estados :

- *ready* : lista de processos prontos à executar. Um processo entra inicialmente na lista de *ready* no momento da sua criação. A transição do estado *ready* para *running* é feito através do escalonamento. A conclusão da primitiva de sincronismo *join*, ou a execução da primitiva *yield*, também são eventos que podem levar um processo do estado *blocked*/*running* para o estado *ready*.
- *running* : processo que está em execução. Um processo em *running* pode passar aos estados *blocked*, *ready* ou terminar. Um processo *running* passa para *ready* sempre que executar uma primitiva do tipo *yield*. Um processo pode passar de *running* para *blocked* através da execução da primitiva *join*.
- *blocked* : processo que estão esperando por um evento, isto é, fim de um outro processo (*join*).

O tipo de escalonamento a ser utilizado é *FIFO NÃO PREEMPTIVO, COM PRIORIDADES*, ou seja, é a política FCFS (First Come, First Served) considerando prioridades SEM preempção. São considerados TRÊS níves de prioridade (baixa, média, alta), sendo que a priodade ALTA é RESERVADA para as funções do μnúcleo (inicialização, escalonador, etc) e não deve ser usada por nenhum processo de usuário.

3. Interface de programação

Para que seja possível o desenvolvimento de programas para o μnúcleo é preciso que este ofereça uma interface de programação (API) para sua chamadas de sistema. As chamadas de sistema a serem implementadas no μnúcleo são descritas nas próximas seções.

3.1 Inicialização

```
int  libsisop_init();
```

libsisop_init serve para preparar o μnúcleo para ser utilizado. É a primeira função que deve ser chamada. Todo e qualquer procedimento de inicialização necessário deve ser feito dentro desta função. A função retorna zero, se corretamente executada, ou outro valor qualquer em caso de erro.

3.2 Gerência de processos

```
int  mproc_create(int prio, void * (*start_routine)(void*), void * arg);
```

mproc_create cria um novo processo a ser executado concorrentemente com o processo chamador desta primitiva. O novo processo executa a função *start_routine* passando *arg* como argumento. O processo termina sua execução ao atingir o fim "normal" da função. O valor retornado por **mproc_create** é o identificador do processo recém-criado (*pid*). Em caso de erro na criação de processo, o valor retornado é -1.

A prioridade (*prio*) é um número inteiro que define, se igual a um (1), que o processo criado terá prioridade MÉDIA. Se for igual a dois (2), o processo criado terá prioridade BAIXA. Qualquer outro valor empregado para prioridade deverá gerar um erro de criação em **mproc_create**.

```
void mproc_yield(void);
```

Um processo pode liberar o processador voluntariamente realizando uma chamada a **mproc_yield**. Neste caso o processo será reinserido no final da lista de process *ready*. ATENÇÃO : a primitiva **mproc_yield** libera voluntariamente o processador APENAS para processos de prioridade igual ou superior ao do processo que está executando. A função **mproc_yield** retorna um valor inteiro com a seguinte interpretação : se o valor retornado for zero, a função foi corretamente executada, caso contrário um valor diferente de zero é retornado.

3.3 Sincronização de término

```
int  mproc_join(int pid);
```

mproc_join suspende a execução do processo corrente até que o processo identificado pelo argumento (*pid*) termine. A função **mproc_join** retorna um valor inteiro com a seguinte interpretação : se o valor retornado for zero, a função foi corretamente executada, caso contrário um valor diferente de zero é retornado.

4. Biblioteca libsisop.a : μnúcleo

As funcionalidades do μnúcleo deverão ser postas a disposição de usuários através de uma biblioteca : a *libsisop.a*. Considerando a existência de dois arquivos fontes C, *arquivo1.c* e *arquivo2.c*, uma biblioteca composta por estes dois arquivos, em ambientes UNIX, é criada através das seguintes linhas de comando :

```
user% gcc -c arquivo1.c -Wall
user% gcc -c arquivo2.c -Wall
user% ar crs libsisop.a arquivo1.o arquivo2.o
```

Faz parte desta solução o fornecimento do arquivo de cabeçalho (*header file*) com os *prototypes* das funções disponibilizadas por *libsisop.a*, ou seja, aquelas descritas na seção 3 deste documento. Este arquivo de inclusão deve OBRIGATORIAMENTE ser nomeado como *unucleo.h* e estar no diretório *include* (ver seção 8).

5. Empregando o μnúcleo : execução e programação

Para facilitar a utilização do μnúcleo supõem-se o seguinte. O "efeito" de um *shell* será substituído por um programa C que deverá ser lançado à partir do *shell* normal do sistema operacional existente na máquina (procedimento padrão de execução de programas). A partir do *main* deste programa C poderão ser lançados n processos através da primitiva de criação de processos. Cada processo corresponderá na verdade a execução de uma função deste programa C. Após a criação de todos os processos, o *main* do programa C lançador deverá passar o controle ao escalonador do sistema (*scheduler*). O controle só retornará ao *main* quando não houver mais nenhum processo usuário (mproc_create) a ser executado. O pseudo-código abaixo ilustra este procedimento para a criação de três processos (*func0*, *func1* e *func2*). Atente para a fato que de dentro de um processo é possível criar tanto quantos processos se queira. No exemplo, o processo *func1* cria o processo *func2*.

```
#include ./include/unucleo.h

void *func0(void *arg) {
     \*corpo da função func0 *\
}

void *func1(void *arg) {
     int id;
     \*corpo da função func1 *\
     id = mproc_create(2, func2, &i);
     . . .
}

void *func2(void *arg) {
     \*corpo da função func2 *\
}
```

```
int main(int argc, char *argv[]) {
    int id0, id1;
    ....
    libsisop_init();
    ....
    id0 = mproc_create(1, func0, &i);
    id1 = mproc_create(2, func1, &i);
    .....
    scheduler();
}
```

Após desenvolver um programa em C, como o fornecido acima, o mesmo deve ser compilado e ligado com a biblioteca que implementa o μnúcleo. A seguinte linha de comando realiza esta etapa :

```
user% gcc -o exemplo exemplo.c -L. -lsisop -Wall
```

Para executar o programa basta então – a partir da linha de comandos – fornecer seu nome :

```
user% exemplo
```

ATENÇÃO : a opção "-L" fornece o caminho no sistema de arquivos onde bibliotecas específicas estão armazenadas. No exemplo acima, o "ponto" indica que é o diretório local de onde *exemplo* está sendo compilado e ligado.

6. Material suplementar de apoio

O μnúcleo definido constitue na verdade o que se chama *biblioteca de threads em nível de usuário, modelo N :1*. Na realidade o que está sendo implementado é uma espécie de máquina virtual que realiza o escalonamento de processos virtuais sobre um processo real do sistema operacional. Na Internet se encontra várias implementações de núcleos similares ao que está sendo solicitado. ENTRETANTO, NÃO SE ILUDAM ! ! NÃO É SÓ COPIAR ! ! Estes códigos são muitos mais completos e complexos do que vocês precisam fazer. Eles servem como uma boa fonte de inspiração.

7. *Road map* para a implementação

Algumas dicas do que precisará ser feito. Primeiro, é preciso definir uma estrutura de dados para representar um processo (o equivalente ao PCB). No PCB estarão todas as informações relativas a um processo (*pid*, estado, contexto, etc). Segundo, deve ser feito rotinas para tratamento de listas encadeadas prevendo inserção e a retirada de elementos. Os elementos da lista são os PCB. As listas implementam o estado *ready* e o estado *blocked*. Terceiro, implementar o escalonador com a política solicitada e o despachante (*dispatcher*). Por fim, será preciso elaborar um conjunto de programas de testes.

IMPORTANTE : O programa deve obrigatoriamente ser implementado em C (não em C++) e executar em ambientes GNU/Linux. O trabalho pode ser desenvolvido em DUPLA (dupla significa "dois alunos").

8. Entregáveis

Deverá ser entregue, via moodle, um arquivo *tar.gz* (**sem** arquivos *rar* ou similares ! !), cujo nome deve ser o número de cartão UFRGS de um dos membros do grupo. O *tar* deve conter os fontes (arquivos.c), os arquivos de inclusão (arquivos .h), arquivo de makefile, programas de teste elaborados pelo grupo (arquivos .c), além da documentação do programa. É obrigatório obedecer a seguinte estrutura de diretórios :

```
\unucleo
    |----makefile  (arquivo de makefile)
    |----src       (onde estarão os fontes do unucleo)
    |----include   (onde estarão os arquivos .h do unucleo, em especial, unucleo.h)
    |----bin       (onde será gerado o executável do unucleo e seus objetos)
    |----testes    (programas de testes elaborados pelo grupo)
    |----lib       (onde estará a biblioteca libsisop.a)
    +----doc       (relatório do programa em formato PDF)
```

Atenção : faz parte da avaliação a obediência RÍGIDA a estes padrões de entrega.

9. Critérios de avaliação

O trabalho será avaliado da seguinte forma :

- **1 ponto :** uso das melhores práticas de programação : clareza e organização do código, programação modular, makefiles, arquivos de inclusão bem feitos (sem código C dentro de um *include* ! !) e comentários "inteligentes". Obediência a especificação (incluir gerar biblioteca, entrega de arquivo *tar.gz*, estrutura de diretórios fornecida na seção 8.
- **1.5 pontos :** documentação. A documentação corresponde a responder o questionário fornecido abaixo.
- **1.5 pontos :** elaboração dos programas de teste (quantidade e qualidade !).
- **6 pontos :** funcionamento do programa de acordo com a especificação. Para seu teste serão empregados programas padrão desenvolvidos pelo professor e pelos programas de teste elaborados pelo grupo.

Atenção : faz parte da avaliação a obediência RÍGIDA a estes padrões de entrega.

Questionário base para documentação

1. Nome dos componentes do grupo e número do cartão.
2. Descrição da plataforma utilizada para desenvolvimento. Qual o tipo de processador (número de cores, com ou sem suporte HT) ? Qual a distribuição GNU/Linux utilizada e a versão do núcleo ? Qual a versão do gcc ? Se o trabalho foi feito ou não em ambientes virtualizados ? Em caso afirmativo, qual a máquina virtual utilizada (versão) ?
3. Para cada programa de teste elaborado pelo grupo : descrever o que programa faz ; indicar claramente quais os parâmetros a serem passados para sua execução e qual a saída esperada.
4. Explique o funcionamento da primitiva *mproc_create* desenvolvida pelo grupo, citando as principais estruturas de dados envolvidas e funções chamadas.
5. Explique o funcionamento da primitiva *proc_yield* desenvolvida pelo grupo, citando as principais estruturas de dados envolvidas e funções chamadas.
6. Explique o funcionamento da primitiva *mproc_join* desenvolvida pelo grupo, citando as principais estruturas de dados envolvidas e funções chamadas.
7. Descrever o que funciona no μnúcleo desenvolvido e o que NÃO está funcionando. Em caso de não funcionamento, dizer qual é a sua visão do porquê deste não funcionamento.
8. Qual a metodologia de teste utilizada ? Isto é, quais foram os passos (e programas) efetuados para testar o μnúcleo desenvolvido ? Foi utilizado um *debugger* ? Qual ?
9. Quais as principais dificuldades encontradas e quais as soluções empregadas para contorná-las.

10. Data de entrega e avisos gerais— LEIA com MUITA ATENÇÃO ! ! !

1. O trabalho pode ser feito em DUPLAS (de dois ! duplas com mais de dois terão sua nota final dividida pelo número de participantes do grupo)
2. O trabalho deverá ser entregue até as 23 :59 :00 horas do dia 03 de MAIO de 2012 via moodle. Entregar um arquivo *tar.gz* conforme descrito na seção 8.
3. Trabalhos entregues atrasados serão penalizados com descontos : Entrega até 10 de maio de 2012 (23 :59 :00 horas) desconto de DOIS pontos ; entrega até 17 de maio de 2012 (23 :59 :00 horas), desconto de QUATRO pontos. Expirado o atraso máximo (em 17/05/2012), nenhum trabalho será mais aceito.
4. O professor da disciplina se reserva, caso necessário, o direito de solicitar uma demonstração do programa com a presença de todo o grupo. A nota final será baseada nos parâmetros acima e na arguição sobre questões de projeto e de implementação feitas ao(s) aluno(s).