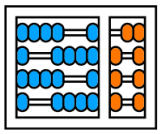

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

MC504A — Sistemas Operacionais
2° Semestre de 2023

Projeto 2

Kernel Calls

Manipulação de processos

Carlos Alberto Astudillo Trujillo (Professor)
Márcio Moraes Lopes (Estudante de Doutorado - PED)

Campinas, Setembro de 2023

1 Resumo

Este projeto contextualiza os processos e o funcionamento do Sistema Operacional Minix. Para isso, espera-se que os estudantes possam manipular os estados dos processos através do kernel do sistema operacional. Adicionalmente, espera-se que durante o desenvolvimento de este projeto o estudante seja exposto a desafios inerentes ao trabalho em sistemas de software (por exemplo, leitura e entendimento de código de terceiros, compilação e execução de sistemas abertos, administração de tarefas de desenvolvimento de software em equipes, entre outras), que desenvolva habilidades de investigação e recopilação de informação, e possa solucionar problemas abertos.

A seguir, as instruções para este projeto.

2 Instruções

2.1 Possíveis correções do uso do Virtual Box no projeto 1

Durante o processo de monitoria do projeto 1, foram constatadas algumas inconformidades quanto ao tamanho do disco. Caso sua VM esteja com apenas os 4GB de HD sugeridos no projeto 1, recomenda-se que seja criada uma nova VM com 10GB de armazenamento.

Siga e execute novamente a instalação do Minix até o passo 24 da Seção 3.1 do Projeto 1. **Obs.: Atente-se para criar uma máquina com 10GB de armazenamento.**

2.2 Trabalho a fazer

1. Este projeto deverá ser desenvolvido em grupos de até quatro estudantes. Se possível, mantenha o mesmo grupo. No entanto, se houver mudanças, por favor comunicar ao PED para as devidas alterações.
2. Leiam detalhadamente as instruções (pode ser necessárias outras leituras para entender o trabalho).
3. O PED ficará disponível para retirar dúvidas quanto ao trabalho. Importante: o PED NÃO VAI FAZER O TRABALHO PARA VOCÊS.
4. Este projeto está dividido em duas grandes etapas:
   - Criação de um serviço que invoque uma kernel call implementada pelo grupo (Seção 3)
   - Manipulação do estado de um processo via kernel call (Seção 4)
5. Devem criar um repositório `projeto-2` no seu do grupo do Gitlab do IC para manter os resultados do seu trabalho.
6. Este repositório deve ser um fork do Minix que usaremos na disciplina (`https://gitlab.ic.unicamp.br/mc504-2023-2/minix`). Ou ainda um fork do projeto 1 de seu grupo.
7. Após a criação do novo repositório, clone o projeto com os seguintes passos:
   a) Entre na pasta /usr
      ```
      cd /usr
      ```
   b) Clone o novo repositório
      ```
      git clone https://gitlab.ic.unicamp.br/mc504-2023-2/grupoXX/projeto-2.git src
      ```
   c) Neste ponto, recomenda-se um snapshot da VM
   d) Compile o novo projeto 2
      ```
      cd /usr/src
      make build
      ```
   e) Neste ponto, recomenda-se um snapshot da VM
8. Haverá a necessidade de entregar um relatório com as atividades desenvolvidas. Os detalhes deste relatório está na Seção 5.

2.3 Entregáveis

Devem entregar no Classroom:

- `relatorio_pojeto2_nome_de_um_integrante_do_grupo.tex`
- `relatorio_pojeto2_nome_de_um_integrante_do_grupo.pdf`

Obs.: Caso o código-fonte do seu relatório tenha vários arquivos .tex (os arquivos criados pelo LaTeXnão devem ser inclusos), devem empacotá-los e subir um arquivo só.

2.4 Data de entrega

A data de entrega é no dia 16 de outubro de 2023. Deverão subir os arquivos no Classroom dentro da tarefa do projeto. Não precisam duplicar as submissões. Uma pessoa por grupo deverá submeter o projeto.

Recomenda-se submeter os trabalhos com antecedência e não esperar aos últimos minutos para a submissão. Não se considerarão correios nem commits enviados depois da data limite (independentemente do motivo do atraso).

3 Criação e invocação de uma kernel call

Esta etapa do projeto está dividida duas partes:

1. Criação de uma chamada ao kernel (kernel call)(Seção 3.1)
2. Criação de um serviço que invoca a kernel call criada (Seção 3.2)

3.1 Criação da kernel call: kpadmon

Você deve criar uma kernel call com o seguinte nome: **kpadmon**

Os passos a seguir foram baseados no seguinte tutorial: https://wiki.minix3.org/doku.php?id=developersguide:newkernelcall.

1. Adicionar o protótipo de nova função do Kernel em: /usr/src/minix/kernel/system.h

```
...
int do_padconf(struct proc * caller, message *m_ptr);
#if ! USE_PADCONF
#define do_padconf NULL
#endif

int do_kpadmon(struct proc *caller, message *m_ptr);
#endif  /* SYSTEM_H */
```

2. Criar do_kpadmon.c que receberá a implementação da sua função em: /usr/src/minix/kernel/system/

```
#include "kernel/system.h"
#include <minix/endpoint.h>
#include <stdio.h>

int do_kpadmon(struct proc *caller_ptr, message *m_ptr)
{
        printf("***********MC504***********\n");
        printf("Oi, eu sou uma Kernel Call\n");
        //implementacao de uma tarefa aqui
        return OK;
}
```

3. Adicionar do_kpadmon.c no arquivo: /usr/src/minix/kernel/system/Makefile.inc

```
...
do_schedule.c \
do_schedctl.c \
do_statectl.c \
do_kpadmon.c
```

4. Criar o mapeamento da função do_kpadmon para SYS_KPADMON em: /usr/src/minix/kernel/system.c

```
...
map(SYS_UPDATE, do_update);           /* update a process into another */
map(SYS_STATECTL, do_statectl);       /* let a process control its state */
map(SYS_KPADMON, do_kpadmon);         /* MC504 kernel call */
```

5. Adicionar o protótipo de sys_kpadmon em: /usr/src/minix/include/minix/syslib.h

```
...
#define PROC_EVENT_SIGNAL       0x02    /* process has caught signal */
int proceventmask(unsigned int mask);
int sys_kpadmon(unsigned flags, endpoint_t proc_ep, message *m_ptr);
#endif /* _SYSLIB_H */
```

6. Repetir o processo do passo anterior 5 para: /usr/include/minix/syslib.h

7. Adicionar o número da nova kernel call (há dois incrementos) em: /usr/src/minix/include/minix/com.h

```
...
#  define SYS_SAFEMEMSET (KERNEL_CALL + 56)     /* sys_safememset() */
#  define SYS_PADCONF (KERNEL_CALL + 57)        /* sys_padconf() */
#  define SYS_KPADMON (KERNEL_CALL + 58)        /* sys_kpadmon() */

/* Total */
#define NR_SYS_CALLS    59      /* number of kernel calls */
```

8. Repetir o processo do passo 7 para: /usr/include/minix/com.h

9. Adicionar a chamada de SYS_KPADMON em: /usr/src/minix/commands/minix-service/parse.c

```
...
{ "MEMSET",             SYS_MEMSET },
{ "PADCONF",            SYS_PADCONF },
{ "KPADMON",            SYS_KPADMON },
{ NULL,          0 }
};
```

10. make e, em seguida, make install em: /usr/src/minix/commands/minix-service

```
cd /usr/src/minix/commands/minix-service
make
make install
```

11. Criar e escrever a implementação da sua função sys_kpadmon.c em: /usr/src/minix/lib/libsys/

```
#include "syslib.h"

int sys_kpadmon(unsigned flags, endpoint_t proc_ep, message *m_ptr)
{
        return (_kernel_call(SYS_KPADMON, m_ptr));
}
```

12. Adicionar o arquivo sys_kpadmon.c ao Makefile em: /usr/src/minix/lib/libsys/Makefile

```
...
sys_vtimer.c \
sys_vumap.c \
sys_kpadmon.c \
taskcall.c \
...
```

13. Construir o Kernel com make includes e depois make hdboot em: /usr/src/releasetools

```
cd /usr/src/releasetools
make includes
make hdboot
```

14. Quando der certo, sugere-se um snapshot aqui

15. commit

3.2 Serviço que invoca a kernel call KPADMON

Deve ser criado um serviço chamado **spadmon** que irá invocar a kernel call recem criada, kpadmon.
Os passos a seguir foram baseados no seguinte tutorial: https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming#example_1hello_world

1. Criar a pasta spadmon em /usr/src/minix/servers

```
cd /usr/src/minix/servers
mkdir spadmon
```

2. Criar o arquivo Makefile em /usr/src/minix/servers/spadmon

```
# Makefile for the padmon sys_call.
PROG=   spadmon
SRCS=   spadmon.c

FILES=${PROG}.conf
FILESNAME=${PROG}
FILESDIR= /etc/system.conf.d

DPADD+= ${LIBCHARDRIVER} ${LIBSYS}
LDADD+= -lchardriver -lsys

MAN=

.include <minix.service.mk>
```

3. Criar o arquivo spadmon.c em /usr/src/minix/servers/spadmon (chamar a função sys_kpamon(...))

```
#include <stdio.h>
#include <stdlib.h>
#include <minix/syslib.h>

int main(int argc, char **argv)
{
    message m; //usado comunicar-se com uma kernel call.
               //Envia e recebe valores diversos
    printf("Hello, SPADMON!\n");
    sys_kpadmon(SYS_KPADMON,SELF,&m);
```

```
    return EXIT_SUCCESS;
}
```

4. Criar o arquivo spadmon.conf em /usr/src/minix/servers/spadmon (atentar para permissão all)

```
service spadmon
{
        system  ALL;            # ALL kernel calls allowed

        ipc
                SYSTEM PM RS LOG TTY DS VM VFS
                pci inet amddev
                ;
    uid 0;
};
```

5. Adicionar a pasta spadmon no arquivo Makefile em: /usr/src/minix/servers

```
.include <bsd.own.mk>

SUBDIR+=        ds input mib pm rs sched vfs vm spadmon

.if ${MKIMAGEONLY} == "no"
SUBDIR+=        ipc is devman
.endif

.include <bsd.subdir.mk>
```

6. make e, em seguida, make install em /usr/src/minix/servers/spadmon

```
cd /usr/src/minix/servers/spadmon
make
make install
```

7. Executar serviço: spadmon

```
minix-service up /service/spadmon
```

8. Quando tudo der certo, sugiro um snapshot aqui

9. commit

3.3 Crie uma tarefa simples

Para entender o correto funcionamento e fluxo de execução de uma kernel call, após a implementação de **kpadmon** e de **spadmon**, crie uma aplicação simples que:

1. Via terminal, entregue parâmetros para o serviço spadmon
2. Spadmon recebe estes parâmetros, faz algum tratamento (por exemplo, conversão de string para int) e envia estes parâmetros para a kpadmon
3. Kpadmon recebe os parâmetros, processa uma tarefa com estes parâmetros e devolve o resultado para spadmon
4. Spadmon recebe e mostra o resultado na tela
    - Exemplo de tarefa simples: Somar dois valores

- Spadmon recebe dois valores, envia para kpadmon realizar a soma, kpadmon devolve o resultado da soma e spadmon mostra o resultado na tela

5. commit

4 Manipulação dos estados de processos: Aplicação em nível usuário

Para esta tarefa, você deverá construir um programa chamado **padmon**, que funcione através do terminal, onde possam ser executados comandos (definidos na Seção 4.1). O programa deve poder entender as sintaxes dos comandos e seus argumentos. Se lhe dirá a padmon que função ou ação deve executar através de um conjunto de opções que lhe serão enviadas.

4.1 Comandos

Seu programa deve gerenciar as seguintes opções:

- **-ps** (process state): Mostra o estado de todos os processos que atualmente se encontram no sistema, utilizando o seguinte formato:

```
---------------------
 PID/EndPoint  State
---------------------
   002           R
   213           S
   989           Z
---------------------
```

  Os estados possíveis dos processos que deve gerenciar são os seguintes:
  - `D`: Ininterrupta (*uninterruptable*) dormindo (usualmente I/O)
  - `R`: Executando ou executável
  - `S`: Interrupta (*interruptable*) dormindo (esperando por algum evento)
  - `T`: Detido
  - `Z`: Zumbi

- **-r <pid/endpoint>** (running): Recebe por argumento o número de processo `<pid/endpoint>`. Este modo muda o processo `<pid/endpoint>` de seu estado atual ao estado executável.

- **-s <pid/endpoint>** (sleep): Recebe por argumento o número de processo `<pid/endpoint>`. Este modo muda o processo `<pid/endpoint>` de seu estado atual ao estado dormindo.

- **-t <pid/endpoint>** (stoped): Recebe por argumento o número de processo `<pid/endpoint>`. Este modo muda o processo `<pid/endpoint>` de seu estado atual ao estado detido.

- **-z <pid/endpoint>** (zombie): Recebe por argumento o número de processo `<pid/endpoint>`. Este modo muda o processo `<pid/endpoint>` de seu estado atual ao estado zumbi.

- **-e <pid/endpoint>** (exit): Recebe por argumento o número de processo `<pid/endpoint>`. Este modo deve encerrar (terminar) o processo `<pid/endpoint>`.

- **-v** (verbose): Por padrão, seus comandos são silenciosos e não mostram nenhuma mensagem (a menos que seja um erro). Caso se deseje mostrar informação deverá de ser enviado uma opção "verbose" que faz ao comando entregar informação do que acontece. Esta flag pode ser misturada com as outras flags e comandos. Por exemplo, as seguintes são opções válidas: `-v -t 123`, `-vt 123`, `-t 123 -v`.

- **-help** (ajuda): Mostra o conjunto de comandos, sua sintaxes, e uma breve descrição de sua funcionalidade.

Seu programa deve detectar, gerenciar e recuperar-se de erros simples. Por exemplo, deve reconhecer quando se ingressa um processo não existente, se o processo se encontra já em naquele estado, argumentos não válidos, entre outros. Podem guiar-se por outros processos que já existem dentro de Minix para manter uma consistência de sua definição. Lembre que deve manter um padrão dos códigos de retorno de seu sistema.

Durante o processo de desenvolvemento desta aplicação, faça commits sempre que julgar necessário, por exemplo: uma funcionalidade específica funciou, commit!

4.2 Integração com a kernel call KPADMON

O programa `padmon` deve comunicar-se com sua contraparte no kernel para poder realizar as tarefas especificadas. Esta contraparte deverá ser implementada com kernel call `kpadmon`. A kernel call `spadmon` deve ser implementada de maneira similar ao administrador de processos ou o sistema virtual de arquivos.

5 Relatório

Devem preparar um relatório de **máximo 6 páginas** de conteúdo (quer dizer que o limite não inclui figuras e referências) utilizando `IEEEtran.cls` (coluna dupla).

O relatório deve conter (pelo menos)

- Resumo
- Descrição das atividades de criação da kernel call kpadmon
- Descrição das atividades de criação do serviço spadmon
- Descrição das atividades da aplicação padmon
- Conclusão (explicar os pontos principais do trabalho)
- Detalhe da divisão do trabalho. Um paragrafo ou tabela detalhando o que cada membro da equipe fez. (Esse detalhe deve concordar com os commits no repositório.)

Se detectado plágio, os envolvidos terão zero na disciplina, e se dará aviso às autoridades correspondentes para que se tomem as sanciones do caso.

6 Dicas

- Revisem a documentação oficial de Minix para resolver problemas. A informação deste documento é um ponto de partida e não uma guia de tudo o que tem que ser feito.
- Estabeleçam metas iniciais e etapas para avançar no projeto. Por exemplo, instalar Minix, logo comunicar-se e instalar programas através de `ssh`, etc. A tarefa pode demorar mais se não são organizados no uso de seu tempo, e se ficam refazendo as mesmas coisas uma e outra vez não estão avançando.
- Criem seu projeto de maneira incremental. E tenham uma maneira de avaliar a funcionalidade de seu código (o script de teste) que permita a vocês saber instantaneamente se seu código não está funcionado.
- Avalie seu sistema constantemente. Escreva um conjunto de provas que possa executar recorrentemente (através da integração continua) para verificar que seu sistema funciona. Avalie cenários exitosos, assim como também aqueles que contenham erros. Por exemplo, você deve ter certeza que uma função que não recebe inteiros dá erro quando recebe um número inteiro.
- **Não comece o projeto dias antes da submissão**. Planejem sua entrega e façam avanços oportunamente.

- Neste projeto será avaliado o uso do repositório e as contribuições individuais. **Se você não tem commits no repositório será penalizado por não poder demostrar uma contribuição no grupo.**
- Se assume que você está familiarizado com o uso de `Makefiles` e técnicas de depuração (introduzidas em disciplinas anteriores). Se não for o caso, deverá aprender estas ferramentas também. O projeto não requere demasiadas linhas de código, mas requere que entenda o Minix e como utilizar as chamadas básicas do sistema.

7 Avaliação

O projeto abrange a avaliação do relatório (escrita, estruturação, redação, etc.) e as tarefas feitas (prática) de maneira conjunta. Serão avaliados o relatório entregue, assim como o trabalho realizado no repositório (através do histórico no repositório). O detalhe da avaliação é apresentado na Tabela 1.

Tabela 1: Avaliação do projeto.

Item	**%**
Relatório	30
KPADMON (código e funcionalidade)	15
SPADMON (código e funcionalidade)	15
PADMON (código e funcionalidade)	30
Conclusões	10