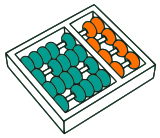

**MC504— Sistemas Operacionais**
INSTITUTO DE COMPUTAÇÃO — UNICAMP
1º Semestre 2018

UNICAMP

# Administração de processos em Minix

## Proyecto 2

## Resumo

Este projeto busca contextualizar ao estudante no Sistema Operacional Minix, em específico, seus processos e funcionamento. Para isso, se espera que os estudantes possam manipular os estados dos processos através do kernel do sistema operacional. Adicionalmente, se espera que durante o desenvolvimento de este projeto o estudante este exposto a desafios inerentes ao trabalho em sistemas de software de grão envergadura (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.

## 1 Administrador de processos

Este projeto consiste em familiariza-los com os modos de operação do sistema operacional e o funcionamento de processos dentro deste. Para tal fim, se pede projetar e implementar uma solução de software que seja capaz de administrar os estados dos processos do sistema operacional.

Sua aplicação contará de duas partes: uma aplicação a nível de usuário e uma a nível de kernel. Ambas partes de seu sistema devem comunicar-se e gerenciar erros que permitam um funcionamento transparente ao usuário. Quer dizer, o usuário não deve saber de que este sistema são duas aplicações que trabalham em sincronia. De tal maneira, deve preocupar-se por gerar um projeto robusto e transparente.

Adicionalmente, como todo projeto de software deve preocupar-se de manter o código de seu projeto através de um administrador de versões, gerar testes unitários, e manter ordenado seu projeto. (Todos estos pontos serão avaliados.) A continuação se descrevem os requerimentos de ambas partes.

## 2 Aplicação nível usuário

O objetivo deste projeto é construir um programa, chamado `padmon`, que funcione através da terminal, onde se possam executar comandos (definidos na seguinte secção). O programa deve poder entender a 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.

### 2.1 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 cerrar (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 bandeira pode ser misturada com as outras bandeiras 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.

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

# 3 Serviço

O programa `padmon` deve comunicar-se com sua contraparte no kernel para poder realizar as tarefas especificadas. Esta contraparte deverá implementar-se como um serviço dentro de Minix, denominado `spadmon`. O serviço `spadmon` deve implementar-se de maneira similar ao administrador de processos ou o sistema virtual de arquivos (se lhe recomenda revisar a implementação destes serviços para poder entender seu projeto).

## 3.1 Configuração de um serviço

A primeira tarefa para a criação de um serviço é gerar uma pasta em `/usr/src/minix/servers/` onde estará todo o código fonte necessário para executar o serviço.

Logo, se deve registrar o serviço na cadeia de compilação do kernel, através da modificação do `Makefile` da pasta `servers` para que seja compilado ao momento de recompilar o kernel, e agregar o serviço à inicialização do sistema, através da modificação do `Makefile` em `/usr/src/releasetools` onde ficam os programas no ordem em que devem ser carregados pelo sistema de inicio.

Note que deverá agregar um rango numérico de memoria que ocuparam os protocolos de petições que realizará o programa `padmon` ao serviço `spadmon` no arquivo `/usr/src/minix/include/minix/com.h`. Ademais, neste arquivo se deve atribuir um `endpoint` que se utilizará para referir-se a `spadmon` ao momento de realizar as chamadas. E também deverá definir o espaço de memoria que estará disponível para o novo serviço em `/usr/src/minix/include/minix/callnr.h`, isto será a base do serviço.

Para que um serviço funcione bem se devem agregar privilégios no sistema. De tal maneira, deve agregar o novo serviço e seus privilégios a `/usr/src/etc/system.conf`.

Finalmente, se deve agregar `spadmon` às tabelas de inicio de `rs`, quem é o serviço encarregado de arrancar e gestionar todos os serviços e tabelas do kernel de Minix em `table.c` de `/usr/src/minix/servers/rs` e `/usr/src/minix/kernel`.

## 3.2 Chamadas a um serviço

Agregar uma nova chamada ao sistema (kernel) de Minix não é difícil. Para poder fazer a chamada deve definir o espaço que estará disponível e as cabeceiras (*headers*) das chamadas no arquivo `/usr/src/minix/include/minix/callnr.h`. Uma vez feito isto, deverá gerar os arquivos necessários dentro da pasta do serviço para definir, implementar e mapear as funções que realizaram as chamadas. Se lhe aconselha revisar os serviços já implementados em Minix para entender como funcionam outros serviços já implementados. Finalmente, para que o programa possa utilizar ditas chamadas deve criar uma biblioteca em `/usr/src/minix/include/minix`.

# 4 Instruções

## 4.1 Trabalho a fazer

1. Este projeto se desenvolverá em trios. No mesmo grupo do projeto anterior. **Caso mude de equipe deve justificar na semana de publicação deste enunciado ao professor.** Senão, se considerará que você continuará na mesma equipe.

2. Leiam detalhadamente as instruções a realizar (pode que precisem mais de uma leitura para entender o trabalho).

3. Precisarão mais informação que os enlaces deste enunciado. Procurem informação enquanto resolvem os problemas. E ainda mais importante, documentem-na para explicá-la no seu relatório.

4. Devem cumprir com os requisitos apresentados nas secções anteriores. Porém, os detalhes de instalação e criação de seu projeto podem mudar, enquanto as interfaces de usuário detalhadas fiquem iguais.

5. Devem criar uma rama `projeto-2` no seu repositório dentro do grupo `mc504/2018s1` do Gitlab do IC para manter os resultados do seu trabalho.

6. Devem modificar ou criar um pipeline para a integração contínua do seu projeto. Devem criar um script de teste que avalie pelo menos uma vez cada uma das características definidas nas secções anteriores.

### 4.2 Entregáveis

Devem entregar no Classroom

- `relatorio.tex`: o código fonte para gerar seu relatório, e
- `relatorio.pdf`: o arquivo de seu relatório.

Caso o código fonte do seu relatório tenha vários arquivos (os arquivos criados pelo LaTeX não devem ser inclusos) devem empacotar eles e subir um arquivo só.

### 4.3 Data de entrega

**A data de entrega é no dia 08 de maio**. Deverão subir os arquivos no Classroom dentro da tarefa do projeto. Não precisam duplicar as submissões. Uma pessoa por grupo deverá submeter o projeto.

Se recomenda 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).

### 4.4 Relatório

Devem preparar um relatório de **máximo 4 páginas** de conteúdo (quer dizer que o limite não inclui figuras e referencias) utilizando `IEEEtran.cls`.

O relatório deve conter (pelo menos)

- Resumo
- Introdução (explicando as ideias principais de Minix usadas no projeto)
- No corpo do trabalho devem desenvolver o projeto (*design*) de sua solução em detalhe (de tal forma que outro desenvolvedor possa duplicar seu trabalho). Este detalhe deve incluir descrições de estruturas de dados, algoritmos utilizados e equações ou fórmulas (no triviais), descrição de cada função relevante (incluindo seu propósito, entradas, saídas, e supostos sobre estes). Deve de explicar o projeto e fundamentação de pelo menos:
  - Projeto e implementação de `padmon`.
  - Projeto e implementação de `spadmon`.
  - Explicação dos comandos (passos) para realizar o serviço, e como este atua sobre os processos.
  - Explicação e resultados sobre as provas unitárias realizadas para validar sua sistema. Justifique porque estas provas demostram que seu sistema cumpre com os requisitos apresentados.
- Conclusão (explicar os pontos principais do trabalho)

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

## 5 Dicas

- Revisem a documentação oficial de Minix para poder 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 da 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). De não ser o caso, deverá aprender estas ferramentas também. O projeto não requere demasiadas linhas de código, mas requere que entenda Minix e como utilizar as chamadas básicas do sistema.

# 6 Avaliação

O projeto contempla a avaliação do relatório (escrita, estruturação, redação, etc.) e as tarefas feitas (prática) de maneira conjunta. Se avaliará o relatório entregue, assim como o trabalho feito no repositório (através do histórico no repositório). O detalhe da avaliação esta dado na Tabela 1.

A parte prática também será avaliada por um script que deverá ser incluso dentro de seu pipeline de integração entregue pelo professor que devera ser executado depois da compilação do kernel e de reiniciar o sistema.

A rubrica será utilizada segundo os quatro critérios definidos por cada dimensão que explicitam os rangos porcentuais que podem ser obtidos segundo o desempeno observado. Excepto na dimensão "escrita técnica" que será avaliada por um único critério.

Além disso, **existe um bônus por entregar scripts de teste que avaliem o código da aplicação** (tanto do cliente como do server). Os códigos que sejam mais criativos e avaliem os casos extremos da aplicação poderão optar por até 10 pontos adicionais. **A entrega dos códigos para obter a bonificação é até o 12 de abril**. Devem enviar os scripts ao correio do professor com o assunto "[MC504] P2: Testes". Depois disso os códigos serão compilados e os mais interessantes terão uma bonificação. Só se dará uma bonificação por grupo. Podem entregar mais de um teste (ou conjunto de testes) para avaliação. Só devem considerar que eles tem que manter o formato de interface descrita neste enunciado e rodar dentro do pipeline de integração continua da disciplina.

**Tabela 1.** Rubrica para a avaliação do projeto.

| **Dimensão** | **Mau**<br>0% | **Bom**<br>1%–50% | **Muito Bom**<br>51%–85% | **Excelente**<br>86%–100% | **Nota** |
|---|---|---|---|---|---|
| ***Relatório*** | | | | | |
| Resumo | Reproduz o enunciado do projeto entregue, ou uma descrição pobre das tarefas realizadas sem explicar pontos principais do trabalho. | Explica a motivação, o problema, e, *quando muito um* dos seguintes, a solução encontrada, os resultados principais, ou a conclusão principal encontrada. | Explica a motivação, o problema, e, *quando muito dois* dos seguintes, a solução encontrada, os resultados principais, ou a conclusão principal encontrada. | Explica a motivação, o problema, a solução encontrada, os resultados principais, e a conclusão principal encontrada. | 2 |
| Introdução | Reproduz o enunciado do projeto entregue, ou uma descrição pobre das tarefas realizadas sem explicar as chamadas ao sistema utilizadas, e não agrega sua visão do problema a resolver. | Explica as chamadas do sistema utilizadas, e o problema a resolver (embora não entra em detalhe); mas não explica como elas se relacionam para resolvê-lo. | Explica as chamadas do sistema utilizadas, e como se relacionam para resolver o problema; mas este não é explicado em detalhe. | Explica a importância das chamadas ao sistema ou enuncia o problema a resolver formalmente, as chamadas do sistema utilizadas, e como se relacionam para resolver o problema. | 4 |
| `padmon` | Não explica o projeto (entradas, saídas, ou lógica de funcionamento) das funções encarregadas de executar `padmon`, sua administração de erros, ou de análises das entradas. | Explica, para que outro programador possa desenvolver usando o código fonte e *com informação adicional*, o projeto (entradas, saídas, e lógica de funcionamento) das funções encarregadas de executar `padmon`, sua administração de erros, e de análises das entradas. | Explica, para que outro programador possa desenvolver usando o código fonte, o projeto (entradas, saídas, e lógica de funcionamento) das funções encarregadas de executar `padmon`, sua administração de erros, e de análises das entradas. | Explica, para que outro programador possa desenvolver *sem informação adicional*, o projeto (entradas, saídas, e lógica de funcionamento) das funções encarregadas de executar `padmon`, sua administração de erros, e de análises das entradas. | 10 |
| `spadmon` | Não explica o projeto (entradas, saídas, ou lógica de funcionamento) das funções encarregadas de executar `spadmon` ou sua administração de erros. | Explica, para que outro programador possa desenvolver usando o código fonte e *com informação adicional*, o projeto (entradas, saídas, e lógica de funcionamento) das funções encarregadas executar `spadmon` e sua administração de erros. | Explica, para que outro programador possa desenvolver usando o código fonte, o projeto (entradas, saídas, e lógica de funcionamento) das funções encarregadas de executar `spadmon` e sua administração de erros. | Explica, para que outro programador possa desenvolver *sem informação adicional*, o projeto (entradas, saídas, e lógica de funcionamento) das funções encarregadas de executar `spadmon` e sua administração de erros. | 10 |
| Conclusões | Apresenta argumentos não relacionados do trabalho desenvolvido, apreciações pessoais (por exemplo, que aprendeu muito no projeto, ou que foi difícil), mas não se desenvolvem argumentos que fundamentno realizado. | Explica as implicações chave ou resultados principais, mas *não entra* em detalhe sobre o argumento principal ou os trabalhos futuros. | Explica, ao menos dois das seguintes, o argumento principal, as implicações chave, ou os resultados principais, e explica com pouco detalhe os argumentos ou trabalhos futuros. | Explica o argumento principal, as implicações chave, resultados principais, repercussões e detalhe dos argumentos apresentados, e o trabalho futuro. | 4 |
| Escrita técnica | Organiza a explicação do desenvolvimento no documento de maneira coerente e clara. Explica o processo de desenvolvimento de maneira coerente e detalhada. Escreve sem faltas de ortografia e redação. (Esta dimensão se analisará num critério único, quer dizer, se ponderará pelo 100%.) | | | | 10 |
| ***Prática***[1] | | | | | |
| `padmon` (grupo) | Executa *ate o 25%* dos comandos, tem excepções e falhas repetidamente. | Executa *mais do 25%* dos comandos sem excepção, é capaz de terminar a execução mas existem erros ou falhas que o detêm. | Executa *mais do 70%* dos comandos sem excepção, é capaz de terminar sem erros ou falhas maiores (quer dizer, `padmon` pode seguir executando-se). | Executa *mais do 85%* dos comandos sem excepção, é capaz de terminar a execução sem erros ou falhas. | 10 |

| **Dimensão** | **Mau**<br>0% | **Bom**<br>1%–50% | **Muito Bom**<br>51%–85% | **Excelente**<br>86%–100% | **Nota** |
|---|---|---|---|---|---|
| `padmon` (individual) | Se avaliará a contribuição individual ao desenvolvimento do componente `padmon` através dos `commit` individuais que se encontrem no repositório oficial da disciplina oportunamente e distribuídos a o longo do tempo (en três ocasiões distribuídas equitativamente dentro do período de desenvolvimento). Quer dizer, a ponderação variará segundo sua contribuição ao projeto segundo os `commit` do repositório nas datas antes mencionadas. | | | | 15 |
| `spadmon` (grupo) | Executa *ate o 25%* das chamadas, tem excepções e falhas repetidamente. | Executa *mais do 25%* das chamadas sem excepção, é capaz de terminar a execução mas existem erros ou falhas que o detêm. | Executa *mais do 70%* das chamadas sem excepção, é capaz de terminar sem erros ou falhas maiores (quer dizer, `padmon` pode seguir executando-se). | Executa *mais do 85%* dos comandos sem excepção, é capaz de terminar a execução sem erros ou falhas. | 15 |
| `spadmon` (individual) | Se avaliará a contribuição individual ao desenvolvimento do componente `spadmon` através dos `commit` individuais que se encontrem no repositório oficial da disciplina oportunamente e distribuídos ao longo do tempo (en três ocasiões distribuídas equitativamente dentro do período de desenvolvimento). Quer dizer, a ponderação variará segundo sua contribuição ao projeto segundo os `commit` do repositório nas datas antes mencionadas. | | | | 20 |
| ***Bônus*** | | | | | |
| Bonificação por testes | Os grupos poderão entregar um conjunto de scripts de teste (que podem ser os mesmos que usarão dentro de sua aplicação ou não) **até o dia 05 de maio**. Aqueles testes que apresentem uma dificuldade considerável e que consigam avaliar as características da aplicação (tanto do server como do cliente) automaticamente, terão uma bonificação.<br>Em caso de dois grupos submeter um teste muito similar (tanto em código como em complexidade para avaliar uma funcionalidade) se dará o bônus ao primeiro em submeter o código.<br>A bonificação é única. Quer dizer, que se um grupo submete mais de um conjunto de instruções de teste que sejam interessantes se dará a bonificação uma vez só (por grupo). | | | | Até 10 |
| | | | | **Total** | **100** |

[1] A execução será avaliada de acordo com os testes executados na integração continua e sua habilidade de passar eles.