PUCRS - Escola Politécnica
Disciplina: Sistemas Operacionais - 2025/1 - Trabalho Prático - Projeto Concorrente - VM
Prof. Fernando Luís Dotti

1. Antecedentes Voce deve contar com o sistema completo, usando paginação e fazendo IO.

2**. Memória Virtual** Nesta fase vamos implementar Memória Virtual, com os seguintes aspectos:

- Ao criar o processo, carrega somente a primeira página em um frame;
- Quando um endereço lógico é utilizado e a página não está em memória, gera *page-fault*;
- **Page-fault:**
  - Pede-se um quadro a mais para o GM - vide abaixo
  - [**]Adiciona mapeamento página/quadro na tabela de páginas do processo
  - Encaminha pedido para trazer pagina do processo ao dispositivo de IO (disco) especifico para paginação
  - Processo executando vai para estado bloqueado
  - Escalona outro processo
- **Pede quadro para gerente de memória**
  - Gerente de memória tenta achar quadro livre
  - Se não achar, tem que vitimar uma página ocupando um quadro
    - Escolher página (escolher uma política simples)
    - Salvar página em disco
      - Encaminha pedido para salvar pagina em disco, possibilitando trazer novamente
    - Processo executando vai para estado bloqueado
    - Escalona outro processo

- **Interrupções novas,** informando:
  - **Fim de salvamento de página em disco**, liberando quadro (so ocorre se alguem pediu este quadro)
    - Passa respectivo quadro para o processo demandante
    - Continua em [**]
  - **Fim de carga de página em quadro alocado da memória**
    - Passa respectivo processo de bloqueado para pronto (fim do tratamento de page-fault)

Desta forma, passamos a ter mais um dispositivo, o Disco:

Contém tanto os programas armazenados como quadros de memória salvos em disco com estado da execução.

Podemos:

- trazer páginas específicas de programas específicos para um quadro de memória
- salvar/copiar páginas específicas, em quadros da memória, para disco (quando a página é vitimada)
- trazer páginas, anteriormente vitimadas, revolta para a memória, em quadro disponível.

Estendendo o esquema de paginação, agora uma página pode:

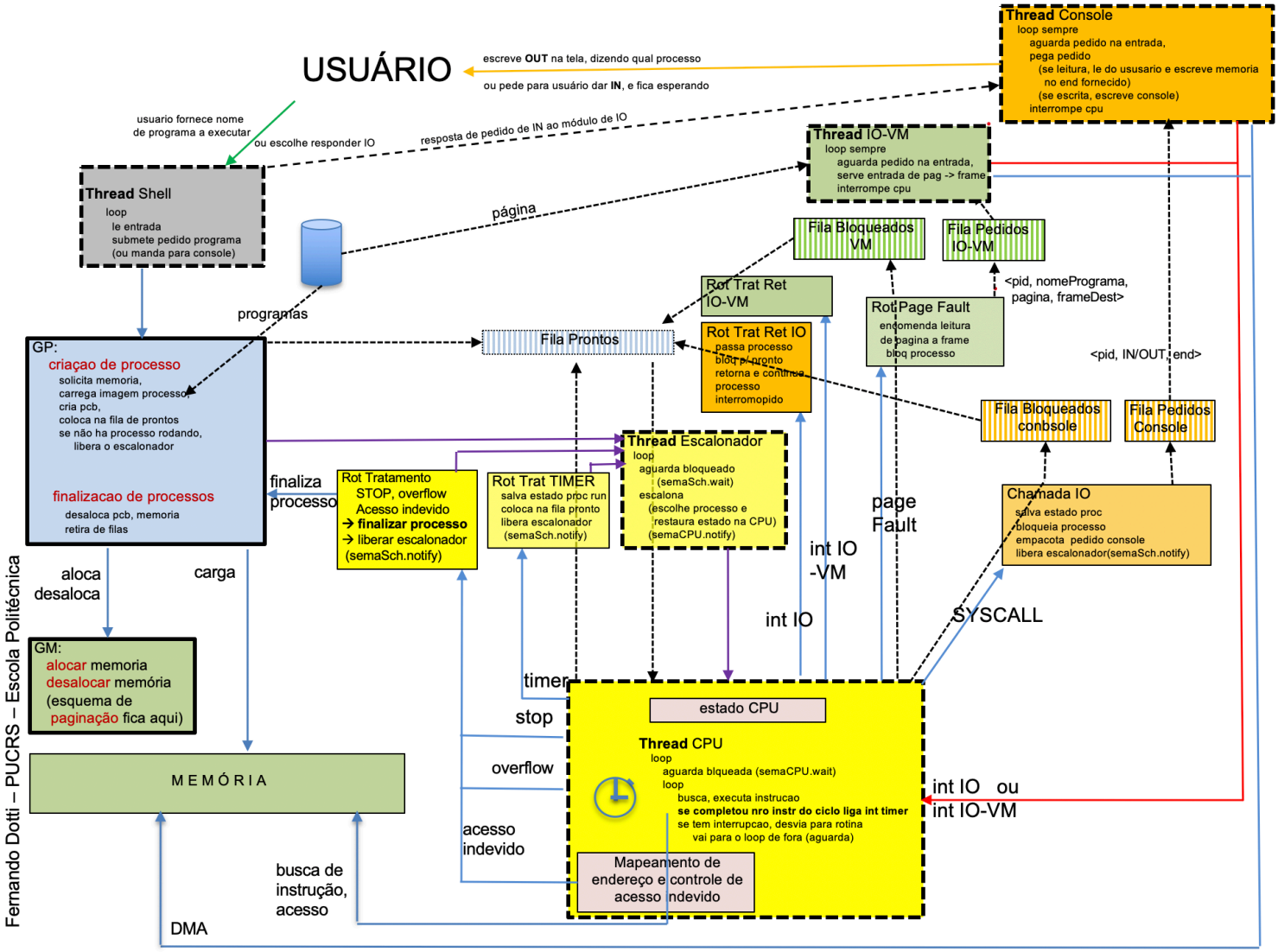
- Nunca ter sido carregada em memória, neste caso o conteúdo é a página do programa original;
- Já ter sido carregada anteriormente, e neste caso o conteúdo está em um quadro de memória ou em um espaço em disco armazenando uma cópia do quadro que esteve em memória.

A informação para diferenciar isto deve estar na tabela de páginas. Você pode implementar o espaço em disco para manter cópias de quadros da memória de forma parecida com o gerente de memória: ele aloca quadros e libera quadros.

Do ponto de vista de estado do processo, ele tem os mesmos 3: **running, ready, e blocked**.

3**. Testes** Os mesmos testes anteriores devem funcionar. Deve ser possível saber os quadros utilizados por um processo e ver a mudança durante a execução.

4**. Esquema**


USUÁRIO
escreve OUT na tela, dizendo qual processo
ou pede para usuário dar IN, e fica esperando
Thread Console
loop sempre
aguarda pedido na entrada,
pega pedido
(se leitura, le do ususario e escreve memoria
no end fornecido)
(se escrita, escreve console)
interrompe cpu
usuario fornece nome
de programa a executar
ou escolhe responder IO
resposta de pedido de IN ao módulo de IO
Thread Shell
loop
le entrada
submete pedido programa
(ou manda para console)
Thread IO-VM
loop sempre
aguarda pedido na entrada,
serve entrada de pag -> frame
interrompe cpu
página
programas
Fila Bloqueados VM
Fila Pedidos IO-VM
<pid, nomePrograma, pagina, frameDest>
Rot Trat Ret IO-VM
Rot Page Fault
encomenda leitura de pagina a frame
bloq processo
GP:
criaçao de processo
solicita memoria,
carrega imagem processo
cria pcb,
coloca na fila de prontos
se não ha processo rodando,
libera o escalonador
finalizacao de processos
desaloca pcb, memoria
retira de filas
Fila Prontos
Rot Trat Ret IO
passa processo
bloq p/ pronto
retorna e continua
processo
interrompido
<pid, IN/OUT, end>
Fila Bloqueados conbsole
Fila Pedidos Console
Thread Escalonador
loop
aguarda bloqueado
(semaSch.wait)
escalona
(escolhe processo e
restaura estado na CPU)
(semaCPU.notify)
finaliza processo
Rot Tratamento
STOP, overflow
Acesso indevido
→ finalizar processo
→ liberar escalonador
(semaSch.notify)
Rot Trat TIMER
salva estado proc run
coloca na fila pronto
libera escalonador
(semaSch.notify)
Chamada IO
salva estado proc
bloqueia processo
empacota pedido console
libera escalonador(semaSch.notify)
page Fault
int IO -VM
int IO
SYSCALL
aloca
desaloca
carga
GM:
alocar memoria
desalocar memória
(esquema de
paginação fica aqui)
timer
stop
overflow
acesso indevido
estado CPU
Thread CPU
loop
aguarda blqueada (semaCPU.wait)
loop
busca, executa instrucao
se completou nro instr do ciclo liga int timer
se tem interrupcao, desvia para rotina
vai para o loop de fora (aguarda)
Mapeamento de endereço e controle de acesso indevido
int IO ou
int IO-VM
MEMÓRIA
busca de instrução, acesso
DMA
Fernando Dotti – PUCRS – Escola Politécnica