Universidade Federal de Minas Gerais
DCC605: Sistemas Operacionais
Trabalho Prático

Cronograma e execução

Execução: em grupo de até dois alunos
Valor: 14 pontos

Paginador de memória

Neste trabalho você irá desenvolver um paginador, um programa para gerenciar quadros de memória física e implementar memória virtual. Seu paginador desempenhará funções similares aos gerenciadores de memória virtual em sistemas operacionais modernos.

Infra-estrutura de memória

O paginador irá funcionar junto de uma infra-estrutura que emula o hardware de memória de um computador moderno. Num computador moderno, a unidade de gerenciamento de memória do processador realiza várias tarefas. A unidade de gerenciamento de memória intermedia todos os acesso à memória.

1. Para acesso a memória não residente, a unidade de gerenciamento de memória causa uma interrupção de falha de página, transfere controle para o tratador de falha de páginas do kernel, e repete a instrução depois que o tratador termina.
2. Para acesso a memória residente que violam proteções da memória, a unidade de gerenciamento de memória causa uma interrupção de falha de página, transfere controle para o tratador de falha de páginas do kernel, e repete a instrução depois que o tratador termina.
3. Para acesso a memória residente autorizados pela proteção da memória, a unidade de gerenciamento de memória traduz o endereço virtual num endereço físico e acessa o endereço físico.

Neste trabalho, a infra-estrutura de memória irá receber interrupções (sinais) de falha de página, e invocar seu paginador para tratar estas falhas. A infra-estrutura é disponibilizada pelo professor e implementada em duas partes.

Gerenciador de memória do usuário [uvm]

O gerenciador de memória do usuário [uvm.a] “conecta” programas à infra-estrutura de memória. Programas que utilizam a infra-estrutura de memória devem primeiro chamar [uvm_create] para informar à infra-estrutura sua existência. A função [uvm_create] inicializa estruturas de dados e um tratador de falha de segmentação [SIGSEGV] para permitir à infra-estrutura gerenciar a memória do processo. A função [uvm_create] também cria soquetes UNIX para permitir comunicação entre o processo e a infra-estrutura de memória.

Após um programa conectar-se à infra-estrutura usando [uvm_create], ele pode alocar memória chamando [uvm_extend]. A função [uvm_extend] funciona como a chamada de sistema [sbrk] e aumenta o espaço de endereçamento do processo em uma página.[1] A função [uvm_extend] retorna um ponteiro para a nova área de memória. A função [uvm_extend] retorna [NULL] e atribui [ENOSPC] a [errno] caso a memória não possa ser alocada ao processo. O processo pode ler e escrever da área de memória retornada por [uvm_extend] normalmente. Memória alocada com [uvm_extend] não deve ser liberada, i.e., programas não precisam chamar [free] nos ponteiros obtidos com [uvm_extend]. Note que apenas a memória alocada com [uvm_extend] é gerenciada pela infra-estrutura; outras áreas de memória utilizadas pelo processo, como a pilha, não são gerenciadas.

Programas podem também pedir à infra-estrutura de memória que grave uma string de texto no log da infra-estrutura chamando [uvm_syslog]. A função [uvm_syslog] funciona como a chamada de sistema [write], recebendo um ponteiro para o string que deve ser impresso e o número máximo de bytes a ser impresso. A função [uvm_syslog] retorna 0 em caso de sucesso. Caso o programa requisite a gravação de string que esteja fora do seu espaço de endereçamento, [uvm_syslog] retorna -1 e atribui [EINVAL] a [errno].

As declarações das funções [uvm_create], [uvm_extend] e [uvm_syslog] estão no cabeçalho [uvm.h]; as implementações estão em [uvm.a].

Listagem 1: Exemplo de programa que utiliza a infra-estrutura de memória [test6.c]

```
int main(void) {
    uvm_create();
    char *page0 = uvm_extend();
    page0[0] = '\0';
    strcat(page0, "hello");
    printf("%s\n", page0);
    uvm_syslog(page0, strlen(page0)+1);
    exit(EXIT_SUCCESS);
}
```

Gerenciador de memória [mmu]

O gerenciador de memória [mmu.a] simula um controlador de memória. Seu paginador irá utilizar funções do gerenciador de memória para controlar alocação e acesso à memória dos processos conectados à infra-estrutura.

[1] O tamanho das páginas de memória na arquitetura do seu computador pode ser obtido usando [sysconf(_SC_PAGESIZE)].

A função [mmu_resident] mapeia a página de memória virtual iniciando no endereço [vaddr], no espaço de endereçamento do processo [pid], no quadro de memória física [frame]. A página será atribuída permissões configuradas em [prot].[2] Esta função permite ao paginador atribuir um quadro de memória física a um processo. Note que [vaddr] deve ser um múltiplo do tamanho de página no sistema.

A função [mmu_nonresident] retira o mapeamento para a página de memória virtual iniciando no endereço [vaddr] no espaço de endereçamento do processo [pid]. A função [mmu_nonresident] permite ao paginador liberar um quadro de memória física previamente alocado a um processo (via [mmu_resident]).

A função [mmu_chprot] atualiza as permissões de acesso do processo [pid] à página de memória virtual iniciando no endereço [vaddr] para [prot]. A função [mmu_chprot] permite ao paginador controlar quando o processo pode escrever e gravar em suas páginas de memória. Antes de chamar a função [mmu_chprot], seu paginador deve ter feito a página de memória residente chamando [mmu_resident].

A função [mmu_zero_fill] preenche um quadro de memória física com o caractere zero. A função [mmu_zero_fill] deve ser chamada pelo paginador antes de permitir a qualquer processo acessar uma página de memória. A chamada a [mmu_zero_fill] evita que um processo possa recuperar informações de outros processos através de memória não inicializada (o que seria uma falha de segurança).

As funções [mmu_disk_read] e [mmu_disk_write] lêem dados em um bloco no disco para um quadro de memória física e de um quadro de memória física para um bloco no disco, respectivamente. As funções [mmu_disk_read] e [mmu_disk_write] permitem ao paginador liberar quadros de memória física movendo dados de processos para o disco. Como num sistema de memória virtual, páginas salvas em disco devem ser recarregadas na memória física quando acessadas por um programa.

Chamadas às funções do gerenciador de memória [mmu] são transmitidas ao gerenciador de memória do usuário [uvm] através de soquetes UNIX. As funções do gerenciador [mmu] só retornam após as ações terem sido implantadas no respectivo processo.

As declarações das funções do gerenciador de memória estão no arquivo [mmu.h]; as implementações estão em [mmu.a].

---

[2] O parâmetro [prot] pode conter os valores [PROT_NONE], [PROT_READ] ou [PROT_READ | PROT_WRITE], declarados em [sys/mman.h].

Paginador de memória

Você irá desenvolver um paginador para controlar quais quadros de memória física estão alocados a cada processo e quais blocos de disco são utilizados para armazenar cada quadro. Seu paginador deve implementar as seis funções descritas no cabeçalho [pager.h].

A função [pager_init] é chamada no início da execução do executável da infra-estrutura de memória para permitir ao seu paginador inicializar quaisquer estruturas de dados necessárias para o gerenciamento de memória.

A função [pager_create] é chamada quando um novo processo é conectado à infra-estrutura. Seu paginador deve inicializar quaisquer estruturas de dados necessárias para alocar e gerenciar memória ao novo processo. A função [pager_destroy] é chamada quando um um programa termina de executar. Seu paginador deve recuperar todos os recursos (quadros de memória física e blocos no disco) alocados ao processo. A função [pager_destroy] é chamada quando o processo já terminou; sua função [pager_destroy] não deve chamar nenhuma das funções do gerenciador de memória [mmu]. Em outras palavras, não é necessário chamar [mmu_nonresident] por que o processo não irá mais acessar sua tabela de páginas; basta atualizar informações dentro do paginador para que ele reutilize quadros que estavam alocados ao processo que terminou.

Seu paginador deve implementar a função [pager_extend], que irá alocar um quadro de memória a um processo. Sua função [pager_extend] deve reservar um bloco de disco para a nova página; caso não existam mais blocos de disco disponíveis, sua função deve retornar [NULL]. Seu paginador deve retornar o endereço virtual da página de memória no espaço de endereçamento do processo. Páginas devem ser colocadas consecutivamente a partir do endereço [UVM_BASEADDR], definido como [0x60000000] em [mmu.h]; esta região de memória não é normalmente utilizada pelo Linux. O espaço de endereçamento de 0x60000000 até 0x600FFFFF representa a memória “virtual” (o [mmu] faz a conversão para os endereços “físicos”).

Seu paginador deve implementar ainda uma função [pager_fault] para tratar falhas de acesso à memória. A função [pager_fault] recebe o identificador do processo e o endereço virtual que o processo tentou acessar. A função [pager_fault] só é chamada para endereços em páginas alocadas por [pager_extend], logo todas as falhas de acesso à memória repassadas a [pager_fault] podem ser tratadas. Sua função [pager_fault] deve utilizar as funções do gerenciador de memória [mmu] para recuperar um bloco livre e permitir acesso ao endereço de memória [vaddr] pelo processo [pid].

Por último, seu paginador deve implementar a função [pager_syslog]. A função [pager_syslog] deve copiar [len] bytes seguindo o endereço [addr] para um espaço de armazenamento

temporário e depois imprimir a mensagem um byte por vez em formato hexadecimal como segue:

```
for(int i = 0; i < len; i++) {        // len é o número de bytes a imprimir
    printf("%02x", (unsigned)buf[i]); // buf contém os dados a serem impressos
}
```

Sua função [pager_syslog] deve comportar-se como se tivesse fazendo leitura no espaço de endereçamento do processo. A função [pager_syslog] deve permitir valores arbitrários para [len] (inclusive maiores que uma página). A função [pager_syslog] deve retornar zero em caso de sucesso e -1 caso o string não esteja contido no espaço alocado por [pager_extend] ao processo.

Paralelismo

Note que seu paginador será chamado sequencialmente pelo sistema de memória. Porém, chamadas podem ocorrer em paralelo, seu paginador deve serializar as requisições para evitar problemas de sincronização, por exemplo, usando mutexes.

Determinismo

Seu paginador deve ter comportamento determinístico, para fins de correção semi-automática. Para isso, sempre que for escolher um quadro de memória livre para alocar a um processo ou um bloco de disco para salvar uma página, escolha o primeiro quadro livre ou o primeiro bloco de disco disponivel.

Adiamento de trabalho

Sempre que possível, seu paginador deve adiar trabalho o máximo para o futuro. Por exemplo, um quadro de memória física não precisa ser alocado ao processo em [pager_extend]. A alocação do quadro de memória física deve ser adiada até o programa acessar a página. De forma similar a função [mmu_zero_fill] só deve ser chamada quando o processo acessa uma página não inicializada. Adie chamadas para todos os métodos do gerenciador de memória [mmu] o máximo possível.

Minimização de trabalho

Quando você tiver mais de uma escolha a respeito do funcionamento do seu paginador, você deve tomar a escolha que reduz a quantidade de trabalho que o paginador tem de fazer. Em particular, uma falha de página (chamada a [pager_fault]) é mais barata do que zerar o conteúdo de um quadro (chamada a [mmu_zero_fill]) que é mais barata do que gravar um quadro no disco (chamada a [mmu_disk_write]).

Política de reposição de páginas

Quando o paginador precisa de um quadro livre para tratar uma falha de acesso em [pager_fault] e não existirem mais quadros livres no sistema, seu paginador deverá liberar um quadro de memória física. Seu paginador deve escolher qual quadro de memória física liberar utilizando o algoritmo da segunda chance. Note que, como a infra-estrutura de memória não mantém informações sobre quais páginas foram acessadas, seu paginador deverá se encarregar de manter esta informação.[3]

Você não precisa criar uma estrutura para armazenar informações de páginas no disco, já que elas são mantidas pelo módulo [mmu] e você poderá acessá-las via funções de acesso ao disco da biblioteca.

Implementação, entrega e avaliação

Utilizando a infra-estrutura de memória

Um pacote com as bibliotecas do gerenciador de memória [mmu.a] e do gerenciador de memória do usuário [uvm.a] estão disponibilizadas no Moodle. O pacote contém ainda os cabeçalhos [uvm.h], [mmu.h] e [pager.h] para permitir você desenvolver programas que usam a infra-estrutura (incluindo [uvm.h] e ligando com [uvm.a]) e seu paginador (incluindo [mmu.h] e ligando com [mmu.a]). Note que a bilioteca [mmu.a] já possui uma função [main], que inicializa a infra-estrutura de memória e chama [pager_init] antes de esperar que programas se conectem à infra-estrutura. Em outras palavras, a compilação deve seguir os seguintes passos:

```
gcc -Wall pager.c mmu.a -lpthread -o mmu
gcc -Wall test6.c uvm.a -lpthread -o test6
```

Implementação do paginador

Seu grupo deve entregar o código fonte do paginador num arquivo [pager.c]. Seu paginador será ligado ligado com o gerenciador de memória [mmu.a] e verificado com uma bateria de testes implementada pelo professor. Para fins de correção automática, seu paginador não deve gerar nenhuma saída além da especificada em [pager_syslog]:

```
printf(“pager_syslog pid %d %s\n”, (int)pid, message);
```

[3] A manutenção de bits de acesso à memória em software (como neste trabalho) é necessária quando o hardware do controlador de memória não provê esta informação.

Bateria de testes

Seu grupo deve entregar também a bateria de testes que utilizou para verificar o paginador desenvolvido. Cada teste deve ser entregue em um arquivo [testX.c] (onde X é um número de até dois dígitos), que será ligado com o gerenciador de memória do usuário [uvm.a] e utilizado para verificar várias implementações erradas de paginadores implementadas pelo professor. Todos os testes entregues pelo grupo irão executar com quatro quadros de memória física e oito blocos de disco. Todos os testes entregues devem também executar em menos de 60 segundos e gerar no máximo 10 MiB de saída, esses limites são mais que suficientes para expor as falhas nas implementações erradas de paginadores implementadas pelo professor.

Testes com múltiplos processos conectados à infra-estrutura devem ser implementados em um único arquivo [testX.c]. Os múltiplos processos devem ser gerados utilizando [fork]. UM teste é considerado completo quando o processo termina de executar; em testes com múltiplos processos, o processo pai deve esperar todos os filhos terminarem de executar.

Relatório

Cada grupo deve preencher e entregar o arquivo [report.txt] incluso no pacote com as bibliotecas.

Pontos extras

Em razão desta ser a primeira edição deste trabalho os alunos que fizerem uma das contribuições abaixo, dentre outras possíveis, serão recompensados com pontos adicionais:

- Melhorias na especificação do trabalho.
- Melhorias na documentação do código.
- Identificação de erros nas bibliotecas da infra-estrutura de memória.