Lista Exercícios 1
MC504A - Sistemas Operacionais
2s 2023
Prof. Carlos A. Astudillo

A continuação apresenta um conjunto de problemas que lhe serão de ajuda para exercitar-se e estudar a matéria. Estes problemas **não têm nota** associada. Porém, se recomenda fortemente resolvê-los.

Introdução a SO

1. O que é um sistema operacional (SO)?
2. Qual a diferença entre monoprogramação e multiprogramação. Exemplifique.
3. O que é um processo? Dê exemplos.
4. Defina as propriedades essenciais dos seguintes tipos de SO: Batch, Multiprogramados, Tempo Compartilhado, Tempo Real e Multiprocessados.
5. O que é uma chamada de sistemas?
6. Explique as principais funções de um SO. De um exemplo de um SO como Árbitro, Ilusionista e Cola.
7. Se o computador possuir apenas um processador (principal), é possível um processamento paralelo ocorrer? Justifique a resposta.

Stack

8. Dado o seguinte código:

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5    float temperature, pressure;
6    cout << gay-lussac(temperature, pressure);
7  }
8
9  float gay-lussac(float t, float p){
10   float k = p/t;
11   return k;
12 }
```

   a. Desenhe o stack antes da execução da linha 6.
   b. Desenhe o stack quando encontra-se executando a função gay-lussac (linhas 9 a 12).
   c. Como é o stack da linha 5 a respeito do stack da execução da linha 6?

9. Dado o seguinte código:

```
1  #include <stdio.h>
2
3  int main(char C, int O){
4    int H;
5    scanf("\%d", &H);
6    if(H >= 2)
7      Wa(H,O);
8    else
9      Ca();
10 }
11
12 char Wa(int H, int O){
13   int Water=H+2+O;
14   return Ca();
15 }
16
17 char Ca(){
18   bool V=TRUE;
19   return 'd';
20 }
```

a. Desenhe o stack quando encontra-se executando a chamada da linha 7.

b. Desenhe o stack quando encontra-se executando a chamada da linha 9.

c. O que acontece no stack quando se executa o condicional (linha 6 e linha 8)?

10. Um processo P1 se executa até que se desencadeia uma interrupção do temporizador. Então, o kernel toma o controle do processador e decide executar o processo P2.

    a. Explique quais passos são feitos para realizar a transferência, causada pela interrupção, de um processo geral desde modo usuário ao modo kernel.

    b. Desenhe o diagrama de estados, em geral, de um processo.

    c. Explique o que acontece na mudança de contexto entre o processo P1 e P2.

    d. Indique o que acontece com os *PCBs* de cada processo durante a mudança de contexto.

    e. O que contém o stack do kernel de cada processo ao finalizar a mudança de contexto?

11. Considere o seguinte código. Suponha que as funções foo e bar são definidas pelo usuário, e que a função SysCall está definida dentro do kernel (com prototipagem dentro de system.h mas definida aqui, convenientemente).

    a. A partir do código anterior, desenhe o stack do processo e o stack do kernel quando se executa a chamada na linha 19.

    b. A partir do código anterior, desenhe o stack do processo e o stack do kernel depois da chamada na linha 20.

    c. Explique a mudança de modo usuário a modo kernel e vice-versa provocado pela chamada ao sistema SysCall.

```c
#include <system.h>
#include <stdio.h>

char foo(float Que, char Zom){
  char holl='a';
  bar(Que+4,holl);
  return Zom;
}

int bar(float Hip, char Oue){
  float Az=50.5 ;
  bool Jack=False ;
  return Az+Hip;
}

int main(float Q){
  float Arr[3] = { 5, 15, 3};
  char pop='p';
  foo(Arr[1], pop);
  SysCall(Arr[2], Arr[0]);
}

bool SysCall(float Lemon, float Juice){
  int x;
  float LJ=0.1;
  for (x=0; x<3; <++)
      LJ=Lemon+Juice;
  bar(LJ, 'D');
  return true;
}
```

Processos

12. Defina três tipos de execução ou ambientes num computador que gerem uma mudança de modo usuário a modo kernel.
13. Considere o seguinte código que está definido em ded.c.

```c
// Definição de ded.c

#include <fildes.h>
#include <stdio.h>

int main(int argc, char** argv){

  // Se abre um arquivo para ler
  int fd = open ("sample.dat", O_RDONLY);
  char buff[20];
  ssize_t bytes_read = read(fd, buff, sizeof(buf));

  // Se gera um char fora do espaço de memoria legal.
  char* erro = (char*) 0x1000;
  *erro = 13;

  int pv = close(fd);
}
```

    a. Explique em detalhe os passos que realiza o sistema operacional para ler os dados do arquivo fd e armazená-los em buff (ver linha 11). (Deve ao menos descrever as mudanças de modo, os buffer que se utilizam para transferir a informação, e possíveis chamadas ao sistema que se desencadeiam.)
    b. A linha 14 resultará numa violação de segmento de memória, explique o que acontece no sistema operacional em este momento.
14. Dado o seguinte código

```
1  void main(int argc, char** argv){
2    int child == fork ();
3    int x = 5;
4    if (child == 0) {
5      x += 5;
6    } else {
7      child = fork();
8      x += 10;
9      if (child) {
10       x +=5;
11     }
12   }
13 }
```

a. Indique quantos processos se criaram ao momento de terminar de executar o if (linha 12).

b. Gere uma rastreamento dos processos anteriores e os valores que adquire a variável x ao longo da execução do código.

15. Dado o seguinte código, suponha que a função getnewprocess(char**) popula o vector que recebe por parâmetro com um novo processo e parâmetros necessários (o primeiro elemento é o nome do programa e os subsequentes são os parâmetros deste).

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int cont=0;
5  void forkthem(int, char**);
6
7  int main (int argc, char** argv){
8    forkthem (4, argv);
9  }
10
11 void forkthem(int n, char** argv){
12   if (n > 0){
13     int child = fork();
14     n = n-1;
15     if(n%2 == 0 && child == 0){
16       // getnewproces enche argv com um novo processo e argumentos
17       getnewproces(argv);
18       if (execvp(*argv, argv) < 0) {
19         printf("*** ERROR: exec failed\n");
20         exit(1);
21       }
22     }
23     forkthem (n);
24   }
25 }
```

a. Explique quais são as funções de fork() e excev(), e explique o que fazem passo a passo para criar um novo processo.

b. Indique quantos processos são gerados ao finalizar a linha 8, e explique por que chegou a esta conclusão.

c. Indique quantos novos programas diferentes são criados por execvp a partir de este código. Por que?

16. Um pipe chamado tofu (que recebe leituras e escritas) é utilizado pelo programa P1 e P2. De tal maneira, P1 recebe como entrada a saída de tofu, e tofu recebe como entrada a saída de P2.
    a. Indique o tipo de relação entre os programas P1 e P2.
    b. Escreva um comando de terminal que representa a situação antes descrita.
17. Um programa P1 envia sua saída como entrada a P2. O programa P2 faz o mesmo com P3.
    a. Indique o tipo de relação entre os programas P1, P2 e P3.
    b. Escreva um comando de terminal que representa a situação.
18. Considere um kernel monoprocessador onde os programas de usuário podem chamar traps utilizando chamadas ao sistema. O kernel recebe e administra interrupções desde os dispositivos de entrada e saída (I/O). Existe a necessidade de administrar secções críticas dentro do kernel?
19. Chamadas a sistema vs. chamadas a procedimentos
    a. Que tão cara é uma chamada a sistema em comparação a uma chamada a um procedimento?
    b. Escreva um programa de prova simples para comparar o custo de uma chamada a sistema (getpid é um bom candidato em UNIX; revise o man page) em comparação com um procedimento. (Nota: cuidado com que o compilador optimize o código das chamadas a procedimentos. Não compile com o optimizador ligado.)
    c. Execute seu experimento em duas arquiteturas distintas se possível (hardware distinto) e reporte os resultados.
    d. Explique as diferenças (se as há) entre o tempo requerido por sua chamada a procedimento e a chamada ao sistema através de uma discussão sobre que faz cada chamada (seja específico).

    (Pode utilizar chamadas ao sistema como (getpid é um bom candidato em UNIX; revise o man page)

20. Quando um sistema operacional recebe uma chamada de sistema desde um programa, uma mudança acontece em direção ao sistema operacional com a ajuda do hardware. Nessa mudança, o hardware estabelece o modo de operação a modo supervisor, chama ao administrador de traps do sistema operacional num endereço especificado pelo sistema

operacional, e permite ao sistema operacional poder retornar ao modo de usuário depois de terminar a administração da trap.

Agora, considere o stack onde o sistema operacional está executando-se quando recebe a chamada ao sistema. Deveria ser este stack diferente ao das aplicações de usuário? ou pode o sistema operacional utilizar o mesmo stack que o programa de usuário? Suponha que o programa está bloqueado enquanto a chamada de sistema retorna.

21. Considere o seguinte código, onde a função signal estabelece uma função a ser executada com a sinal que se envia por parâmetro.

```
1  void foo(int sig) {
2    fprintf(stderr, "Tentando\n");
3  }
4
5  void bar(int sig) {
6    fprintf(stderr, "Ciao\n");
7    exit(0);
8  }
9
10 void zoo(int sig) {
11   fprintf(stderr, "Tentei\n");
12   signal(SIGINT, bar);
13 }
14
15 void main() {
16   /* SIGINT é quando se pressiona CTRL-C */
17   signal(SIGINT, foo);
18   signal(SIGALRM, zoo);
19   // Envia SIGALRM em 3 segundos depois de ser chamada
20   alarm(3);
21   while (1) {
22     printf("zzz...\n");
23     sleep(1);
24   }
25 }
```

   a. Explique o que faz a função *alarm* quando o tempo estabelecido nela termina. Detalhe o que acontece no computador, as mudanças no espaço de usuário e do kernel, e as partes envolvidas.
   b. Como implementaria a função *alarm*?
   c. Depois de 3 segundos de iniciado o programa anterior, o que imprimirá se CTRL-C é pressionado?
   d. Durante os primeiros 3 segundos de iniciado o programa anterior, o que imprimirá se CTRL-C é pressionado?

Threads

22. Threads

a. Nome e defina os estados de um thread e dê exemplos de situações onde um thread mude de um estado a outro.
b. Explique o que é o TCB e seu conteúdo.
c. Explique passo a passo a mudança de contexto de um thread, ademais especifique que acontece com o TCB do thread.

23. Dado o seguinte código:

```
1  static void go(int n);
2
3  #define N 3
4  static thread_t threads[N];
5
6  int main(){
7    for(int i=0; i<N; i++)
8      thread_create(&threads[i], go, i);
9
10   for(int i=0; i<N; i++){
11     int exitValue = thread_join(threads[i]);
12     printf("Thread %d returned %d \n", i, exitValue);
13   }
14 }
15
16 void go(int n){
17     printf("Hello from thread %d\n",n);
18     thread_exit(100+n);
19 }
```

a. Explique qual é a função de thread_create().
b. Explique qual é a função de thread_join().
c. Durante a execução do código anterior, qual é a quantidade máxima de threads que se estão executando antes de obter a mensagem Thread 1 returned?
d. Durante a execução do código anterior, qual é a quantidade mínima de threads que se estão executando antes de obter a mensagem Thread 2 returned?
e. Qual é a saída esperada da execução das linhas 8 e 11?

24. Dado o seguinte código:

```
1  #define N 3
2  static thread_t threads[N];
3  float global_length=4;
4  float global_width=5;
5
6  float area(float w){
7    float r=global_length*w;
8    global_length--;
9    return r;
10 }
11
12 float foo(float l){
13   return l+l+l+l;
14 }
15
```

```
16  float vol(float h){
17    float v=global_length*global_width*h;
18    global_length--;
19    global_width--;
20    return v;
21  }
22
23  int main(float high){
24    float length=3, width=2;
25    thread_create(&threads[0], area, width);
26    thread_create(&threads[1], foo, length);
27    thread_create(&threads[2], vol, high);
28  }
```

a. Desenhe um esquema de como se vê a memória do processo com os threads.

b. Desenhe o stack do processo e dos threads.

c. Defina o que é uma condição de corrida, e determine se no código anterior existe uma.

d. Se existe uma condição de corrida no código anterior, é gerado um erro quando se executa o programa?

25. Considere o seguinte código. Suponha que as funções foo e bar são definidas pelo usuário, e que a função SysCall está definida dentro do kernel (com um protótipo dentro de system.h mas definida aqui, convenientemente).

    a. Explique qual é a funcionalidade a diferença de chamar a thread_yield() e sleep() num thread.

    b. Especifique passo a passo a mudança a modo supervisor (ou modo kernel) gerado quando um thread chama a sysCall.

    c. Desenhe o stack do processo, kernel e threads.

    d. Comente se existe condição de corrida. Se não, explique por que não existe.

```c
#include <system.h>
#include <stdio.h>

#define N 2
static thread_t threads[N];

char foo(float Que){
  char holl='a';
  bar(Que+4);
  sleep(15);
  return Zom;
}

int bar(float Hip){
  char Oue='a';
  float Az=50.5 ;
  bool Jack=False ;
  SysCall(Az);
  thread_yield();
  return Az+Hip;
}

int main(float Q){
  float Arr[3] = { 5, 15};
  char pop='p';
  thread_create(&threads[0], foo,Arr[0]);
  thread_create(&threads[1], bar, Arr[1]);
}

// prototipada em system.h mas definida acá
bool sysCall(float Lemon){
  int x;
  float Juice=4;
  float LJ=0.1;
  for (x=0; x<3; <++)
      LJ=Lemon+Juice;
  return true;
}
```

26. Dado o código:

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>

#define N 2
static thread_t threads[N];

void escrever(){
  // Se abre um arquivo para escrever
  int fd = open ("sample.dat", O_WRONLY|O_CREAT|O_TRUNC, 0700);
  write(fd, "Escrever aqui algo interessante\n", 36);
  close(fd);
}

void ler(){
  // Se abre um arquivo para ler
  int fd = open ("sample.dat", O_RDONLY);
  char buff[20];
  ssize_t bytes_read = read(fd, buff, sizeof(buf));
  close(fd);
}

int main (int argc, char** argv){
  thread_create(&threads[0], escrever, NULL);
  thread_create(&threads[1], ler, NULL);
}
```

a. Quando um thread executa *read()* se desencadeia uma interrupção e se deve executar código privilegiado. Neste caso, só o thread que executa a chamada se detém enquanto o resto dos threads seguem executando-se com normalidade? Justifique sua resposta.

b. Explique em detalhe os passos que realiza o sistema operacional quando o thread tenta ler os dados do arquivo *fd* e armazená-los em *buff* (ver linha 20). (Deve ao menos descrever as mudanças de modo, os buffer que se utilizam para transferir a informação, e possíveis chamadas ao sistema que se desencadeiam.)

c. Comente se existe uma condição de corrida, em caso de não ter, explique.

27. Dado o seguinte código:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <system.h>
4  #include <pthread.h>
5
6  #define N 2
7  static thread_t threads[N];
8
9  void dynamite(char** argv){
10   int child = fork();
11   if(child == 0)
12       execvp(*argv, argv)
13 }
14
15 void fire(char** argv){
16   char *str;
17   str = (char *) malloc(15);
18   excevp(*argv, argv);
19 }
20
21 int main (int argc, char** argv){
22   int fd = open ("sample.dat", O_RDONLY);
23   thread_create(&threads[0], fire, argv);
24   thread_create(&threads[1], dynamite, argv);
25 }
```

a. Explique que acontece passo a passo quando os threads chamam às funções *fork()* e *excev()*.

b. Se um thread chama a *fork()*, duplica o filho todos os threads do pai? Justifique sua resposta.

c. Se um thread chama a *execv()*, substituirá o programa especificado ao processo completo? Justifique sua resposta.

d. O código anterior é um coquetel do caos, enquanto a programar de maneira segura com threads se refere. Comente todas as condições de corrida existentes e os possíveis problemas gerados.

e. Explique o que acontece com o arquivo aberto fd no processo filho. Que operações o processo filho pode realizar sobre o arquivo? Que informação é compartilhada ou não entre pai e filho a respeito do arquivo?

Sincronização

28. A classe Impressora imprime documentos utilizando sua função imprimir(Buffer b) que recebe como parâmetro um Buffer para obter os documentos. Os documentos são colocados num Buffer através de uma classe Biblioteca com um método armazenar(Documento d, Buffer b) que recebe como parâmetro um Documento que deve ser armazenado no Buffer.

    Uma implementação preliminar é a seguinte:

```
1  Impresora::imprimir(Buffer b) {
2    Documento d= b.obterDocumento();
3    procesar(d); // aqui procesamos o documento na impresora
4  }
5
6  Biblioteca::armazenar(Documento d, Buffer b) {
7    b.insertarDocumento(d); // insertamos
8  }
```

    Porém, ao executá-la com um Buffer compartilhado temos problemas com ele.

    a. Por que não funciona o código?
    b. Como podemos resolvê-lo?

29. Suponha que no problema anterior a classe Impressora pode administrar até 5 impressoras simultaneamente. Esta nova versão da classe Impressora encapsula a nova funcionalidade através da função *processar*. Porém, a solução anterior não permite utilizar todas as impressoras simultaneamente.
    a. Por que?
    b. Como pode resolver o problema para permitir utilizar várias impressoras simultaneamente?

30. Numa barbearia há um barbeiro que pode trabalhar quando há clientes na barbearia. Porém, quando não há clientes este descansa.

    A barbearia está projetada com um conjunto limitado de cadeiras para que os clientes possam esperar quando o barbeiro está ocupado com um cliente.

    Quando o barbeiro termina de cortar o cabelo a um cliente, este o despacha e vai à sala de espera a chamar ao seguinte cliente. Em caso de que não haja ninguém esperando o barbeiro regressa a descansar.

Cada cliente quando chega à barbearia observa o que acontece na barbearia. Se o barbeiro está descansando, o cliente o acorda e o barbeiro se põe a trabalhar. Do contrário, se senta numa cadeira, se há espaço, senão sai da barbearia sem seu corte de cabelo.

Implemente duas funções, barbeiro e cliente, que simulem o comportamento descrito. (Pode utilizar um comentário para descrever onde faz trabalho o barbeiro e onde espera o cliente por seu corte de cabelo.)

31. Para a seguinte implementação de uma transferência atômica, explique se funciona, não funciona, ou se é perigosa (quer dizer, às vezes funciona e às vezes não). Explique por que e como resolveria o problema em caso de existir.

```
1  void atomicTransfer (queue *queue1, queue *queue2) {
2    item thing; /* o que transferimos */
3    queue1->lock.acquire();
4    thing = queue1->dequeue();
5    if (thing != NULL) {
6      queue2->lock.acquire();
7      queue2->enqueue(thing);
8      queue2->lock.release();
9    }
10   queue1->lock.release();
11 }
```

32. Dada uma primitiva de sincronização por hardware chamada int32 xchg(int32* lock, int32 val) que recebe um ponteiro à variável a trocar e um valor, e que retorna o valor anterior da variável lock. Se a função xchg não consegue acessar à variável lock de forma atômica, ela retornará o valor anterior nela.
    a. Implemente um mutex usando xchg, *i.e.*, implemente as funções *acquire* e *release* do mutex. Explique suas decisões de projeto.
    b. É possível usar xchg para criar um semáforo? Caso possa, escreva uma implementação de um semáforo usando xchg. Explique suas decisões de projeto. Pelo contrário, explique por que não é possível.
33. Você foi contatado pela mãe natureza para ajudá-la com a reação química que gera água, a que parece não funcionar por problemas de sincronização na *matrix*. O truque é obter dois átomos H e um átomo O todos ao mesmo tempo. Os átomos na *matrix* são threads. Cada átomo H invoca um procedimento hReady quando está pronto para reaccionar, e cada átomo O invoca um procedimento *oReady* quando está pronto. Para este problema, a mãe natureza tem o seguinte código que tenta resolver o problema. Supostamente, os procedimentos hReady e oReady esperam que existam dois átomos H e um O, posteriormente um desses procedimentos chama à função makeWater para criar a água. Depois da chamada a makeWater duas instâncias de hReady e uma instância de oReady deveriam retornar. A solução deve evitar esperar por sempre (starvation) e a espera ocupada (busy wait).

Os semaforos da matrix implementam uma política FIFO dentro deles.

a. Explique se o seguinte código funciona, não funciona ou é perigoso (funciona algumas vezes e outras não). Em caso de que não funcione entregue uma possível solução.

```
1  int numHydrogen = 0;
2  semaphore pairOfHydrogen(0); // init 0
3  semaphore oxygen(0);         // init 0
4
5  void hReady() {
6    numHydrogen++;
7    if ((numHydrogen % 2) == 0) {
8      pairOfHydrogen->V();
9    }
10   oxygen->P();
11 }
12
13 void oReady() {
14   pairOfHydrogen->P();
15   makeWater();
16   oxygen->V();
17   oxygen->V();
18 }
```

b. A mãe natureza entrega outra possível solução para você. Explique se o seguinte código funciona, não funciona ou é perigoso (funciona algumas vezes e outras não). Em caso de que não funcione entregue uma possível solução.

```
1  semaphore hPresent(0);     // init 0
2  semaphore waitForWater(0); // init 0
3
4  void hReady() {
5    hPresent->V();
6    waitForWater->P();
7  }
8
9  void oReady() {
10   hPresent->P();
11   hPresent->P();
12   makeWater();
13   waitForWater->V();
14   waitForWater->V();
15 }
```

34. Suponha que tem dois funções que representam um produtor e um consumidor:

```
1  void producer {
2    while(1) {
3      item = produce();
4      queue.enqueue(item)
5    }
6  }
7
8  void consumer {
9    while(1) {
10     item = queue.dequeue(item);
11     consume(item);
12   }
13 }
```

O código supõe que existe uma filha compartilhada *queue* e que as funções têm acesso a ela.

a. O código tem secções críticas? Identifique-as.
b. Usando as variáveis de exclusão mútua, proteja as regiões críticas. O que acontece com o código anterior? Ele funciona como é esperado? Explique sua resposta.
c. Se você pode usar variáveis de sincronização gerais, proteja as regiões críticas. O que acontece com o código anterior? Ele funciona como é esperado? Explique sua resposta.