Sistemas Operativos, Teste 2

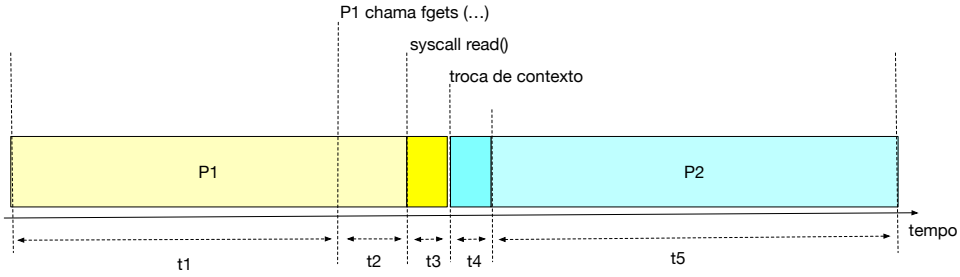
IST - LEIC-A/ LEIC-T/ LETI - 2017-2018
29 de Novembro de 2017

- Todas as respostas devem ser dadas na folha de resposta.
- Não pode sair da sala antes de passarem 60 minutos. Não é autorizada a utilização de telemóveis ou outros dispositivos electrónicos.
- O teste tem a duração de 1 hora.

---

1 Estrutura do Sistema Operativo

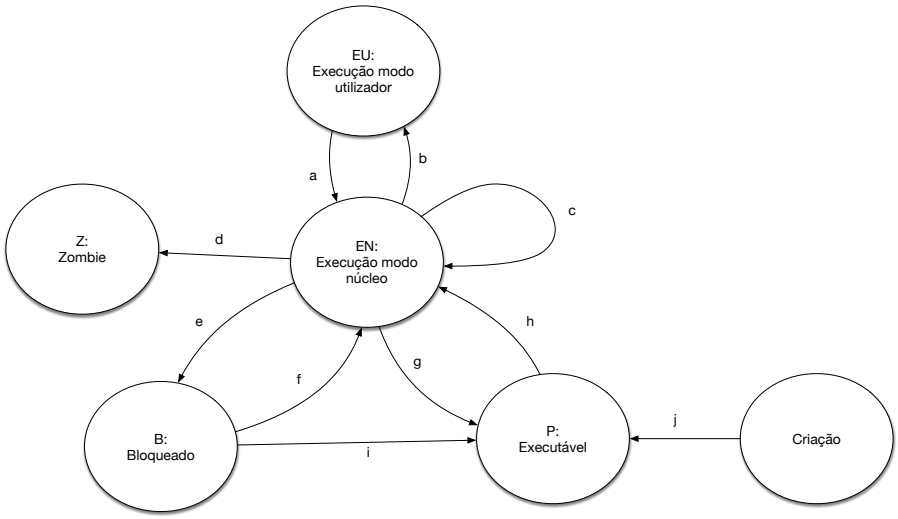
Considere a seguinte figura que ilustra uma sequência de dois processos (P1 e P2) que se executam num único processador. Nesta sequência, o processo P1 faz uma chamada bloqueante e perde o processador.



Pergunta 1 (*2 valores*) Do ponto de vista das intruções máquina usadas para chamar a função de biblioteca fgets() e chamada sistema read() existe alguma diferença? Justifique.

§

Considere agora o seguinte diagrama de transição de estados de um processo:



Pergunta 2 (*1 valor*) Uma das transições representadas na figura está errada e não existe na realidade. Qual é? Justifique.

Pergunta 3 (*2 valores*) Considere de novo a figura com a execução dos processos P1 e P2 acima. A figura captura diversos instantes de tempo diferente ($t1$, $t2$, ..., $t5$). Indique, para cada um destes instantes, qual o estado dos processos P1 e P2, usando os estados que estão representados no diagrama de transição de estado (EU, EN, B, etc).

2 Criação de Processos

Considere o seguinte programa (em todos os programas os *#include* e as verificações dos valores de retorno das funções foram omitidos para simplificar a listagem).

```
main () {
  pid_t mypid = getpid();
  pid_t pidfilho;

  pidfilho = fork();
  if (pidfilho) {
    wait (NULL);
    printf ("Filho %d do pai %d terminou\n", pidfilho, mypid);
    exit (0);
  }
  else {
    mypid = getpid();
    pidfilho = fork();
    if (pidfilho) {
      wait (NULL);
      printf ("Filho %d do pai %d terminou\n", pidfilho, mypid);
      exit (0);
    }
    else {
      mypid = getpid();
      printf ("Foi para isto que o processo %d viveu\n", mypid);
      exit (0);
    }
  }
}
```

Em todas as perguntas seguintes, considere que o valor mypid é igual ao seu número de aluno e que cada processo novo que é criado na máquina obtém um identificador que é igual ao último identificador usado, incrementado de uma unidade (assuma que não há outros programas a lançar processos na máquina).

Pergunta 4 (*2 valores*) Qual o é o output deste programa?

Considere o seguinte programa, cujo binário se denomina `do_stuff`.

```
int global;

main () {
  pid_t mypid = getpid();

  printf ("Processo %d começou a executar\n", mypid);
  global = 100;
  sleep (10);
  printf ("No processo %d global = %d\n", mypid, global);
  printf ("Processo %d vai fazer exit\n", mypid);
  exit (0);
}
```

e o seguinte programa (chamado `do_exec`) que usa o binário `do_stuff`:

```
int global;

main () {
  pid_t mypid = getpid();
  pid_t pidfilho;

  global = 10;
  pidfilho = fork();
  if (pidfilho) {
    wait (NULL);
    printf ("Filho %d do pai %d terminou\n", pidfilho, mypid);
    printf ("No processo %d global = %d\n", mypid, global);
    exit (0);
  }
  else {
    mypid = getpid();
    execl ("./do_stuff", "do_stuff", NULL);
    printf ("Foi para isto que o processo %d viveu\n", mypid);
    exit (0);
  }
}
```

Pergunta 5 (*2 valores*) Considere que a chamada execl não dá erro. Qual o output deste programa?

3 Identificadores de Utilizadores e Permissões

Considere um sistema unix em que cada processo possui associado um "real user identifier" (ruid) e um "effective user identifier" (euid).

Considere que os binários da alínea anterior, `do_exec` e `do_stuff`, possuem as seguintes protecções:

Cenário A
`-rwxr-xr-x 1 luis staff 8680 Nov 14 19:46 do_exec`
`-rwxr-xr-x 1 luis staff 8600 Nov 14 19:46 do_stuff`
Cenário B
`-rwxr-xr-x 1 luis staff 8680 Nov 14 19:46 do_exec`
`-rwxr--r-- 1 luis staff 8600 Nov 14 19:46 do_stuff`
Cenário C
`-rwsr-sr-x 1 luis staff 8680 Nov 14 19:46 do_exec`
`-rwxr--r-- 1 luis staff 8600 Nov 14 19:46 do_stuff`

Pergunta 6 (*3 valores*) Considere que o utilizador david, do grupo de utilizadores staff, decide executar o programa do_exec. Para cada um dos cenários acima, indique se o programa do_exec chega a ser executado, se o programa do_stuff chega a ser executado e, em caso afirmativo, qual o uid e euid dos processos pai e filho no momento em que terminam.

§

4 Sinais

Considere a seguinte variante do programa do_exec, chamada do_exec_kill:

```
main () {
  pid_t mypid = getpid();
  pid_t pidfilho;

  pidfilho = fork();
  if (pidfilho) {
    sleep (1);
    printf ("Processo %d vai matar o processo %d\n", mypid, pidfilho);
    kill (pidfilho, SIGINT);
    wait (NULL);
    printf ("Filho %d do pai %d terminou\n", pidfilho, mypid);
    exit (0);
  }
  else {
    mypid = getpid();
    execl ("./do_stuff", "do_stuff", NULL);
    printf ("Foi para isto que o processo %d viveu\n", mypid);
    exit (0);
  }
}
```

Pergunta 7 (*2 valores*) Considere que a chamada execl não dá erro. Qual o output deste programa?

§

Considere que alterava o programa do_stuff da seguinte forma:

```
void no_way_jose () {
  printf ("Nice try buddy\n");
  signal (SIGINT, no_way_jose);
}


main () {
  pid_t mypid = getpid();

  signal (SIGINT, no_way_jose);
  printf ("Processo %d começou a executar\n", mypid);
  sleep (10);
  printf ("Processo %d vai fazer exit\n", mypid);
  exit (0);
}
```

Pergunta 8 (*2 valores*) Considere que a chamada execl não dá erro e que o processo que corre do_stuff se executa até sleep(10) antes do pai chamar a funcão kill. Qual será o output de executar o processo do_exec_kill?

5 Escalonamento

Considere um sistema operativo no qual:

- Os processos podem estar num de três estados possíveis: em execução (E), executáveis (P), ou bloqueados (B).
- Para simplificar, assuma que quando um processo está a executar só pode fazer uma de duas operações: computar (E_c) ou chamar uma primitiva bloqueante denominada wait (E_w).
- O escalonador usa apenas 5 prioridades, no intervalo $[0, 4]$ em que 0 é a prioridade máxima. A prioridade associada a cada processo é *fixa*.
- Os processos executáveis estão organizados num conjunto de listas, uma lista por prioridade. O núcleo mantém também uma lista separada para os processos que estão bloqueados. Em qualquer caso em que um processo que estava fora da fila de executáveis respetiva retorne a essa fila (porque estava em execução e perdeu CPU, ou porque deixou de estar bloqueado), é colocado no fim da mesma.
- Os processos podem executar (comp e wait) ou bloquear-se por x unidades de tempo (após executarem wait(x)). Quando um processo executa wait(x) num determinado instante, sai de execução no instante seguinte, retornando para a fila de executáveis x unidades de tempo depois.

  Por exemplo, se no instante 1 um processo executar wait(1), ele ficará bloqueado no instante 2, voltando depois para a fila de executáveis na transição para o instante 3; podendo (dependendo do escalonador) executar-se a partir do instante 3 (isto é, já estará executável no início do instante 3, quando o escalonador escolhe a quem atribui o porcessador nesse período). A chamada comp(x) é uma abreviatura para x operações de computação.

Considere que no início do instante 0 existem 4 processos executáveis, cada um com a prioridade indicada na tabela abaixo. Entre os processos com igual prioridade, a ordem dos mesmos na fila é P_2, P_3, P_4. A tabela mostra também as instruções que cada processo executará quando tiver o CPU:

Processo	P_1	P_2	P_3	P_4
Prioridade	1	3	3	3
Instruções	wait(1) comp(1) wait(100)	comp(3) wait(1) comp(3) wait(1) comp(3)	comp(4) wait(1) comp(4) wait(1) comp(4)	comp(1) wait(1) comp(1) wait(1) comp(1)

Considere que, a partir do estado inicial definido na tabela acima, se observou a seguinte evolução dos estados dos processos:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P_1	$\mathbf{E}_w$	B	P	P	P	$\mathbf{E}_c$	$\mathbf{E}_w$	B	B	B	B	B	B	B	B
P_2	P	$\mathbf{E}_c$	$\mathbf{E}_c$	$\mathbf{E}_c$	$\mathbf{E}_w$	B	P	P	P	P	P	P	P	P	$\mathbf{E}_c$
P_3	P	P	P	P	P	P	P	$\mathbf{E}_c$	$\mathbf{E}_c$	$\mathbf{E}_c$	$\mathbf{E}_c$	$\mathbf{E}_w$	B	P	P
P_4	P	P	P	P	P	P	P	P	P	P	P	P	$\mathbf{E}_c$	$\mathbf{E}_w$	B

Pergunta 9 (*1 valor*) Este escalonador é preemptivo? Justifique referindo uma transição no escalonamento acima que prove a sua resposta.

§

Pergunta 10 (*2 valores*) Complete a tabela, dizendo qual o estado de cada processo com o evoluir do tempo, assumindo que o escalonador é preemptivo, usa prioridades fixas com "*round-robin*" e um *timeslice* de 3 unidades.

§

Pergunta 11 (*1 valores*) Considere que P_4 é um processo interativo, ao contrário de P_2 e P_3. Mantendo apenas 5 níveis de prioridade, não alterando o valor do *time-slice* e assumindo que os processos lançados pelos utilizadores são sempre criados com a prioridade 3 (antes de se saber se são interactivos ou "*cpu-bound*"), proponha uma modificação ao escalonador que permita que processos como P_4 melhorarem o seu tempo de resposta aos comandos dos seus utilizadores.

Sistemas Operativos, Teste 2, 29 de Novembro de 2017

IST - LEIC-A/ LEIC-T/ LETI - 2017-2018

Folha de Respostas (1/2)

Número: **Nome:**

§

Estrutura do Sistema Operativo

Pergunta 1	
Pergunta 2	

Pergunta 3		t1	t2	t3	t4	t5
	P1	EU				
	P2	P				

§

Criação de Processos

Pergunta 4	1	.
	2	
	3	
	4	
	5	
	6	
	7	
	8	
Pergunta 5	1	.
	2	
	3	
	4	
	5	
	6	
	7	
	8	

Folha de Respostas (2/2)

Número:
Nome:

§

Identificadores de Utilizadores e Permissões

Pergunta 6

Cenário	Programa	Executa (sim/não)	ruid	euid
A	do_exec			
	do_stuff			
B	do_exec			
	do_stuff			
C	do_exec			
	do_stuff			

§

Sinais

Pergunta 7

1	.
2	
3	
4	
5	
6	
7	
8	

Pergunta 8

1	.
2	
3	
4	
5	
6	
7	
8	

§

Escalonamento

Pergunta 9

Pergunta 10

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P_1	$\mathbf{E}_w$	B													
P_2	P	$\mathbf{E}_c$													
P_3	P	P													
P_4	P	P													

Pergunta 11