MO806/MC914
Tópicos em Sistemas Operacionais
2s2007

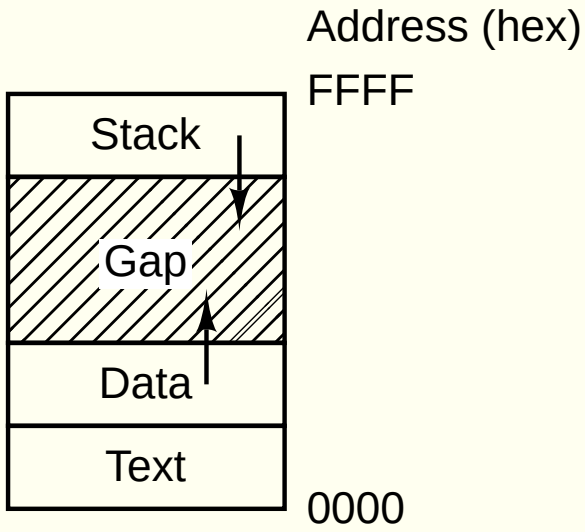
Processos e Threads 1

Objetivos

- Espaços de endereçamento
- Pacote Pthread
- Operações create, join e exit
- Envio e recepção de valores para threads
- Primeiros problemas de condição de corrida

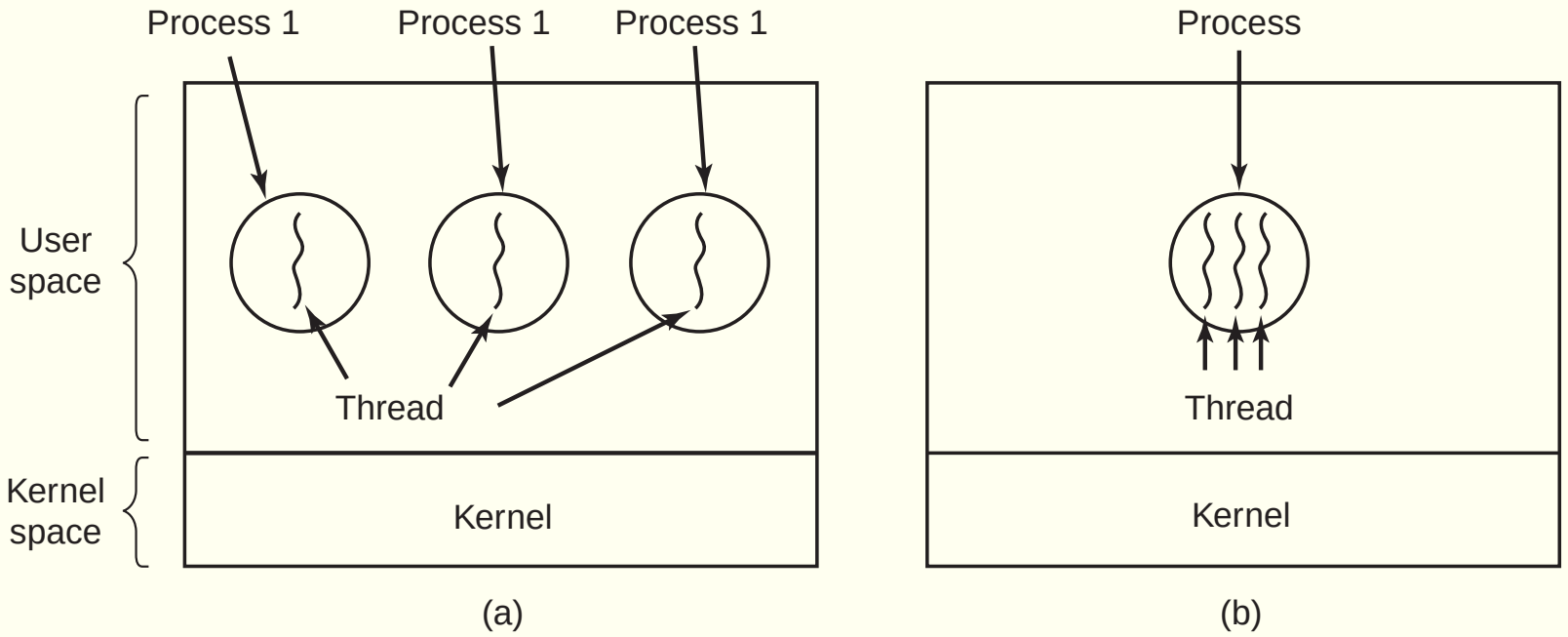
Processo

- Programa em execução
- Espaço de endereçamento

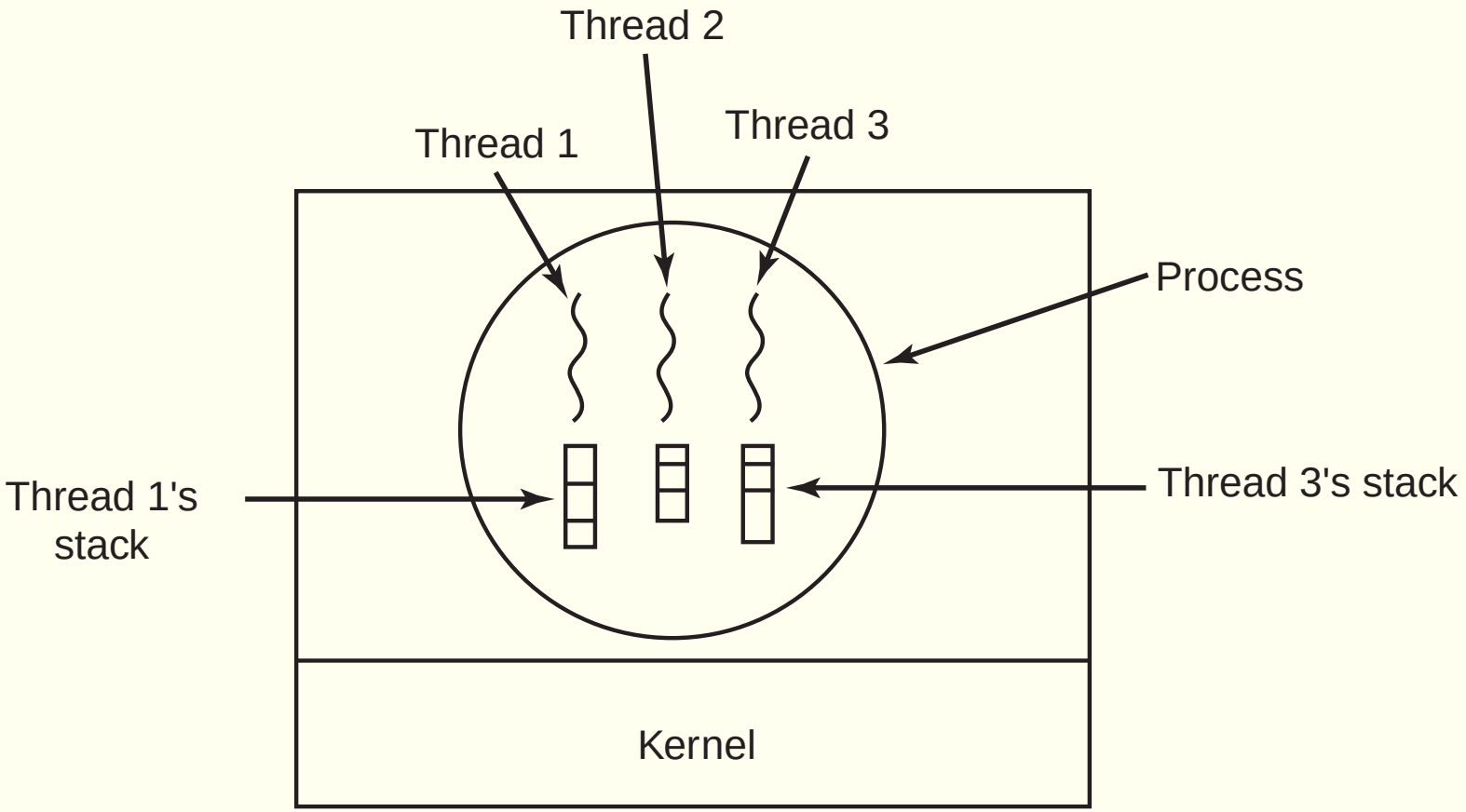


- Veja os códigos: ender.c e ender-malloc.c

Modelo de threads



Pilhas independentes



Como trabalhar com threads

Veja os comandos:

- `pthread_create`
- `pthread_join`
- `pthread_exit`

Como criar uma thread

```
int  pthread_create(pthread_t  *thread,
                    pthread_attr_t *attr,
                    void * (*start_routine)(void *),
                    void *arg);
```

Veja o código: create0.c

Como esperar por uma thread

```
int pthread_join(pthread_t th,
                 void **thread_return);
```

Veja os códigos: join0.c, join1.c, join2.c, join3.c e join4.c

Como passar argumentos para uma thread

- Exemplo: cada thread pode precisar de um identificador único.
- Veja os códigos: create1.c, create2.c, create3.c e create4.c

Como encerrar a execução de uma thread

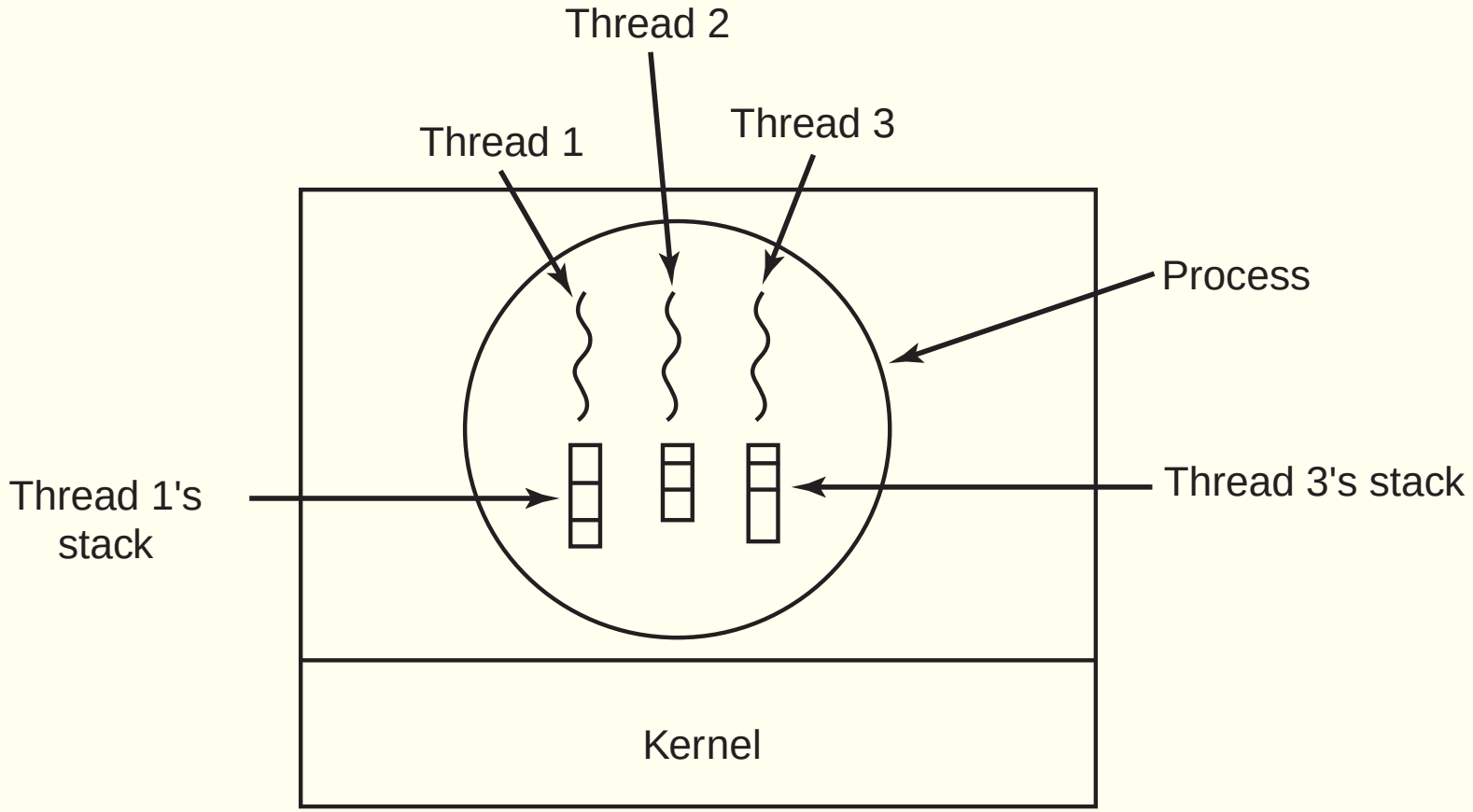
- Comando `return` na função principal da thread (passada como parâmetro em `pthread_create`)
- Análoga ao comando `return` na função `main()`

Como encerrar a execução de uma thread

- `void pthread_exit(void *retval);`
- Análoga ao comando `exit(status);`

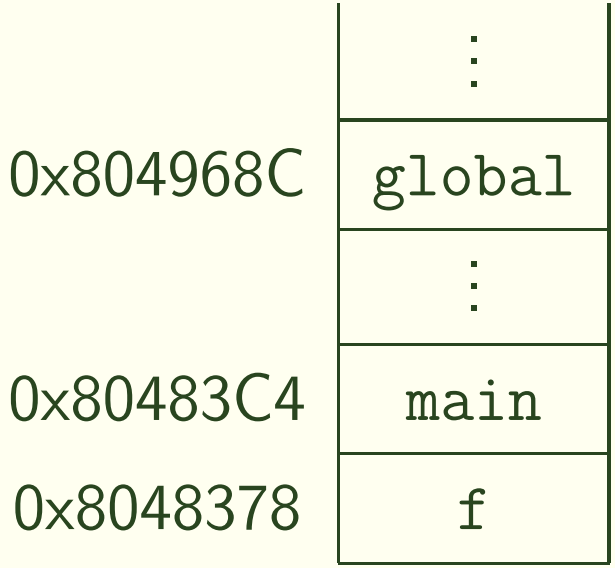
Veja o código: exit0.c

Pilhas independentes

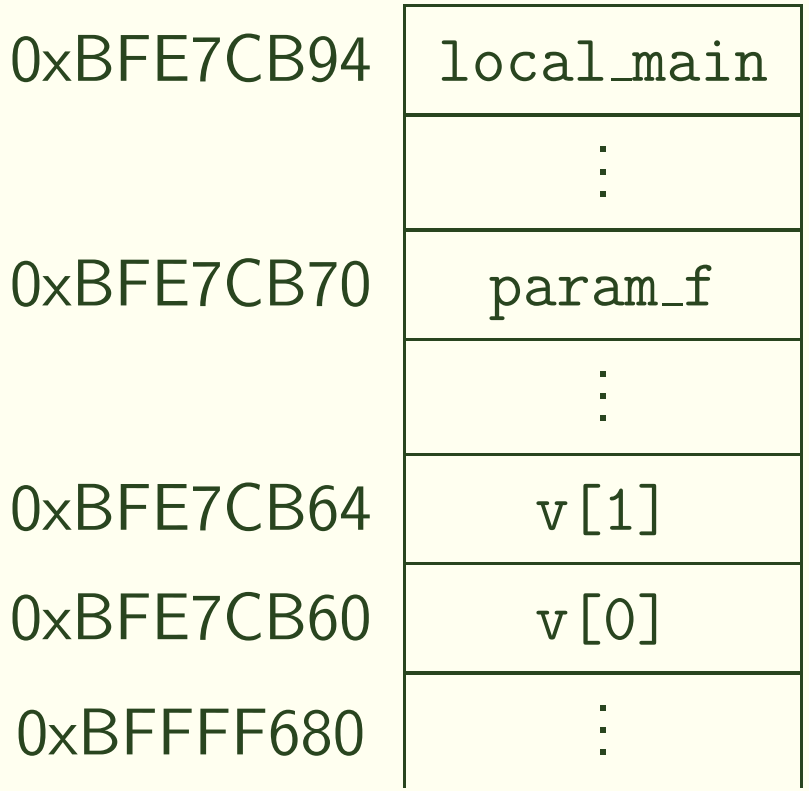


Analise o código pilha0.c para ver os endereços das pilhas.

Exemplo de endereços



Exemplo de endereços



O que pode estar armazenado na pilha?

- Espaço para valor de retorno da função
- Argumentos
- Endereço de retorno
- Registradores
- Variáveis locais

É muito fácil corromper a pilha

- Basta fazer acesso a posições não alocadas de um vetor;
- Veja o código: corrompe_pilha.c

Uma thread pode corromper a pilha de outra thread

- Pilhas são independentes, mas não protegidas
- Veja o código: corrompe_thread.c

Acesso a recursos compartilhados

- Estudo de caso:

```
volatile int s; /* Variável compartilhada */

/* Cada thread tentar executar os seguintes
   comandos sem interferência.   */

s = thr_id;
printf ("Thr %d: %d", thr_id, s);
```

Condição de disputa

Saída esperada

```
int s; /* Variável compartilhada */
```

Thread 0

(i) `s = 0;`

(ii) `print ("Thr 0:  ", s);`

Thread 1

(iii) `s = 1;`

(iv) `print ("Thr 1:  ", s);`

Saída: Thr 0: 0

Thr 1: 1

Condição de disputa

Saída esperada II

```
int s; /* Variável compartilhada */
```

	Thread 0		**Thread 1**
(iii)	`s = 0;`	(i)	`s = 1;`
(iv)	`print ("Thr 0:  ", s);`	(ii)	`print ("Thr 1:  ", s);`

Saída: Thr 1: 1

Thr 0: 0

Condição de disputa

Saída inesperada

```
int s = 0; /* Variável compartilhada */
```

	Thread 0		**Thread 1**
(i)	`s = 0;`	(ii)	`s = 1;`
(iii)	`print ("Thr 0:  ", s);`	(iv)	`print ("Thr 1:  ", s);`

Saída: Thr 0: 1

Thr 1: 1

Veja o código: `inesperada.c`