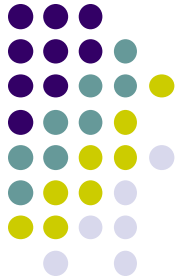
Sistemas de Computação

Threads

Threads



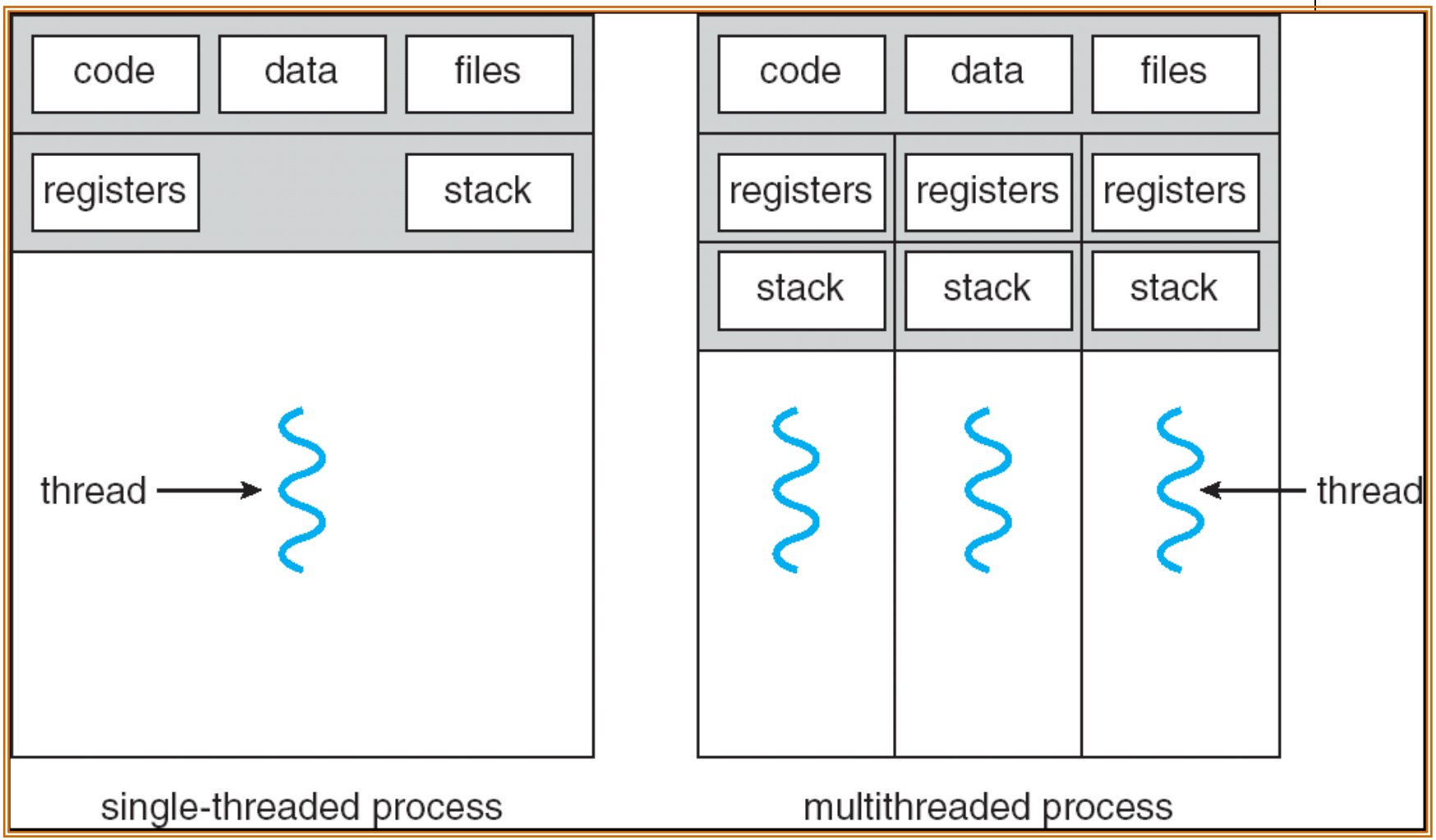
Thread = linha de execução independente dentro de um mesmo processo

Multiplas threads: associação de múltiplos fluxos de trabalho a um processo.

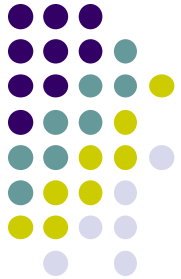
Em certas aplicações é conveniente disparar várias threads dentro do mesmo processo (programação concorrente).

Ex: quando diversos (>1) pedidos de E/S devem ser tratados concorrentemente, e que precisam compartilhar algumas estruturas de dados (e.g. uma cache em um servidor de arquivos ou conexões TCP em um servidor Web)

Processos com 1 ou mais treads



Principais Características



- Cada thread tem a sua pilha própria, mas compartilha o mesmo espaço de endereçamento do processo em que foi criada;
- Se duas threads executam o mesmo procedimento/método, cada uma terá a sua própria cópia das variáveis locais;
- As threads podem acessar todas os dados globais do programa, e o heap (memória alocada dinamicamente)
- Nesse acesso a dados globais (i.e. quando o acesso inclui mais do que uma instrução de máquina), as threads precisam ter acesso em regime de exclusão mútua (p.ex. usando locks( ))

Exemplo de uso de threads

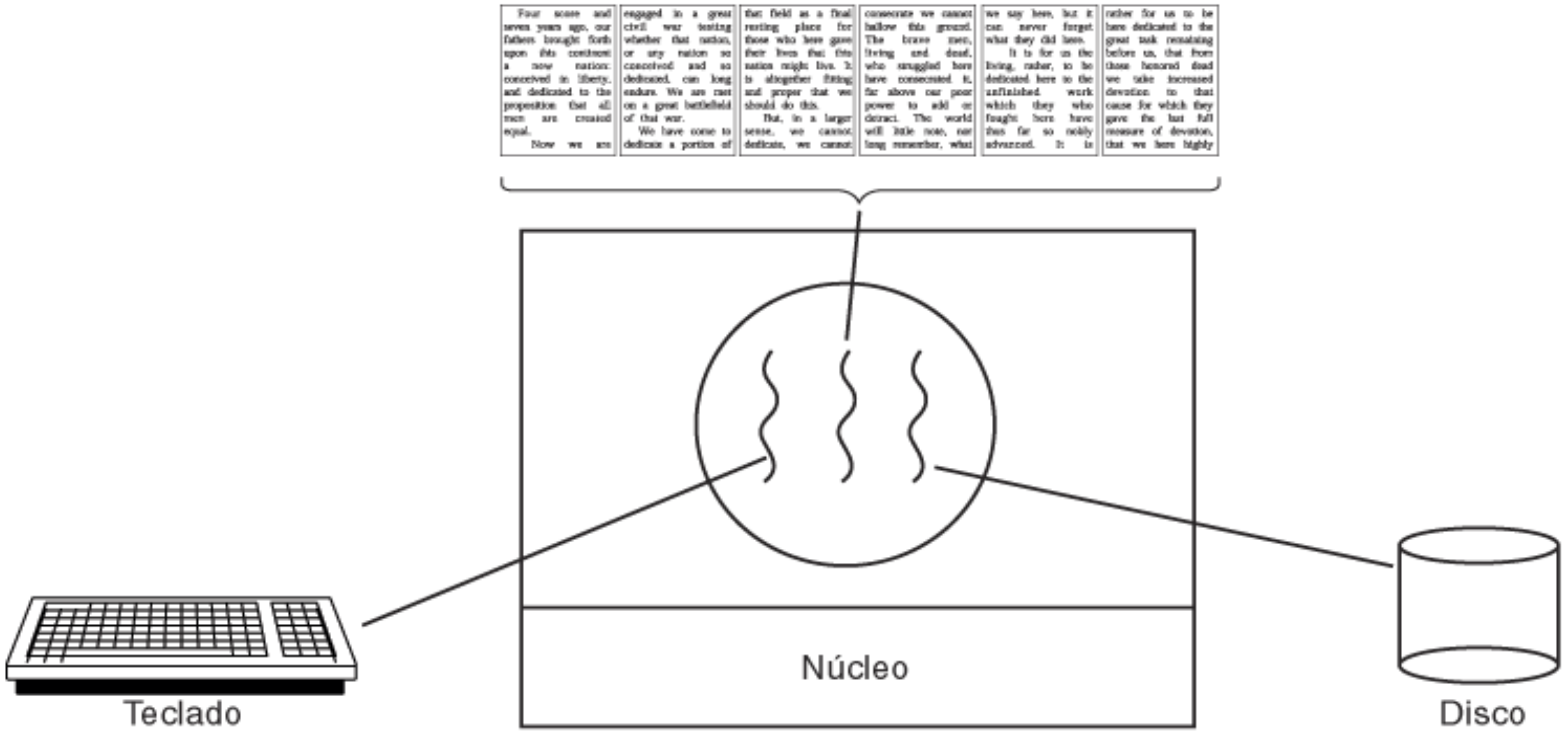
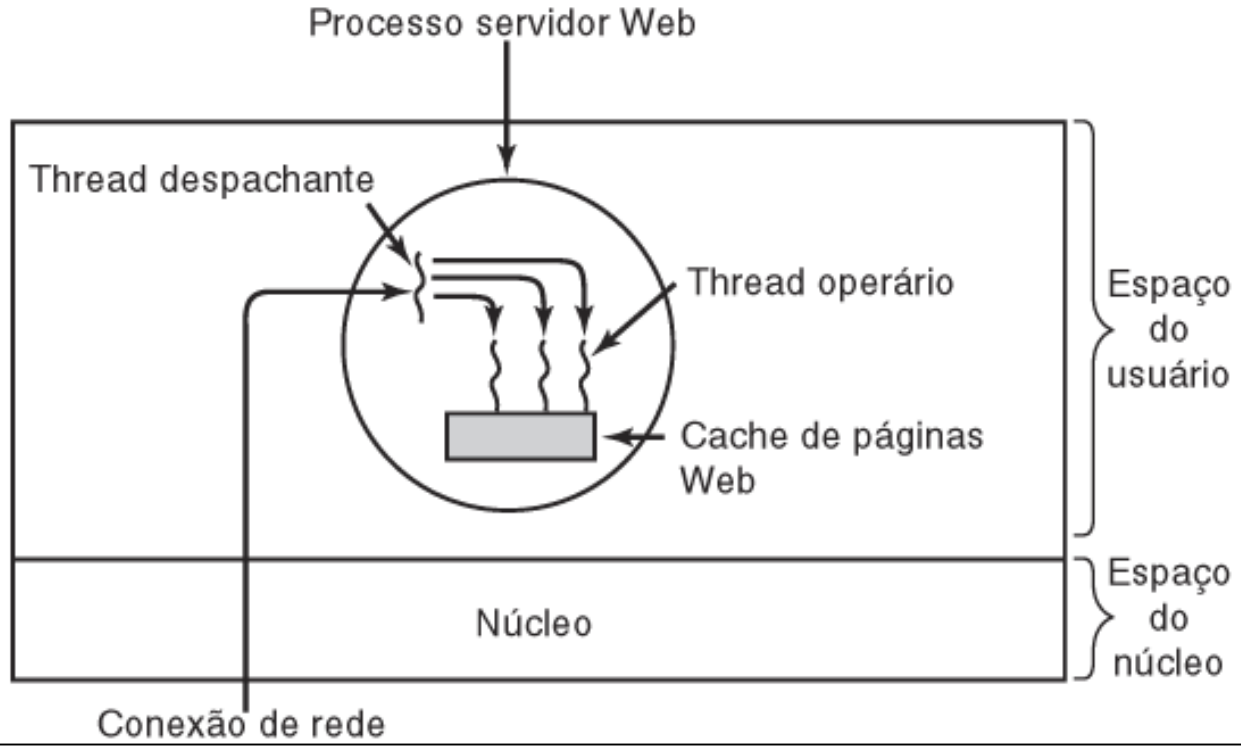


Fig.: Um processador de texto com três threads

Exemplo de uso de Threads

Um servidor web com múltiplas threads



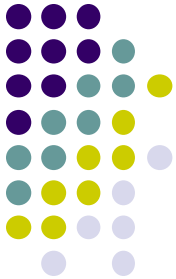

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

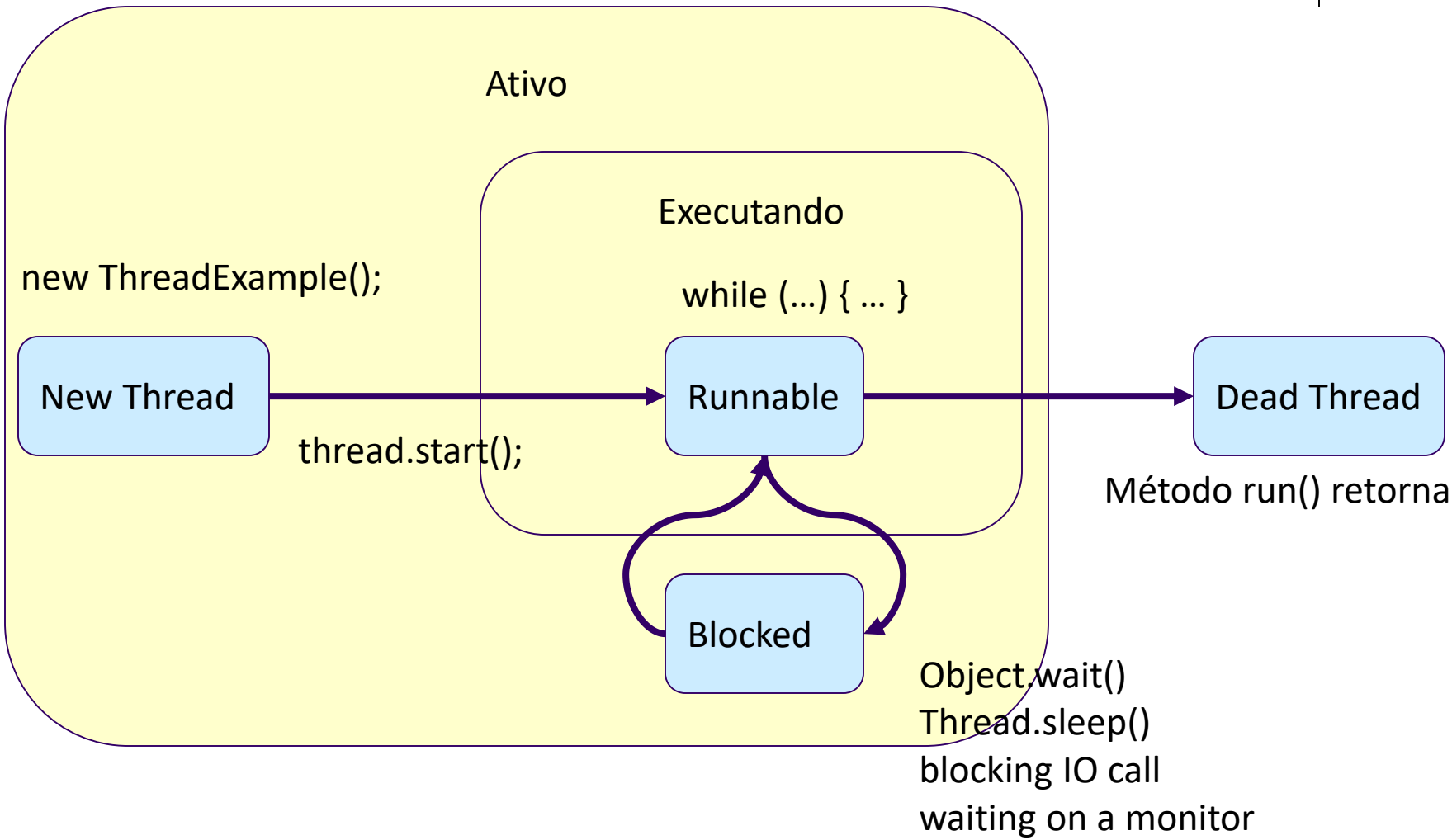
(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

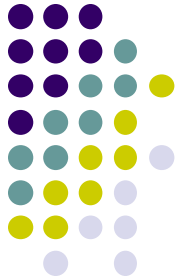
(b)

Diagrama de estados de threads





Gerenciamento de Threads

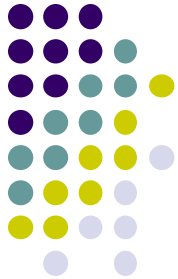


Ao contrário de processos, threads compartilham a mesma região de memória

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- Cada thread possui sua própria pilha e contexto de CPU (conteúdo de PC, SP, registradores, PSW, etc.)
- A troca de contexto entre threads é mais leve do que a troca de contexto entre processos
- Uma tread pode estar nos estados: *running, blocked & ready*

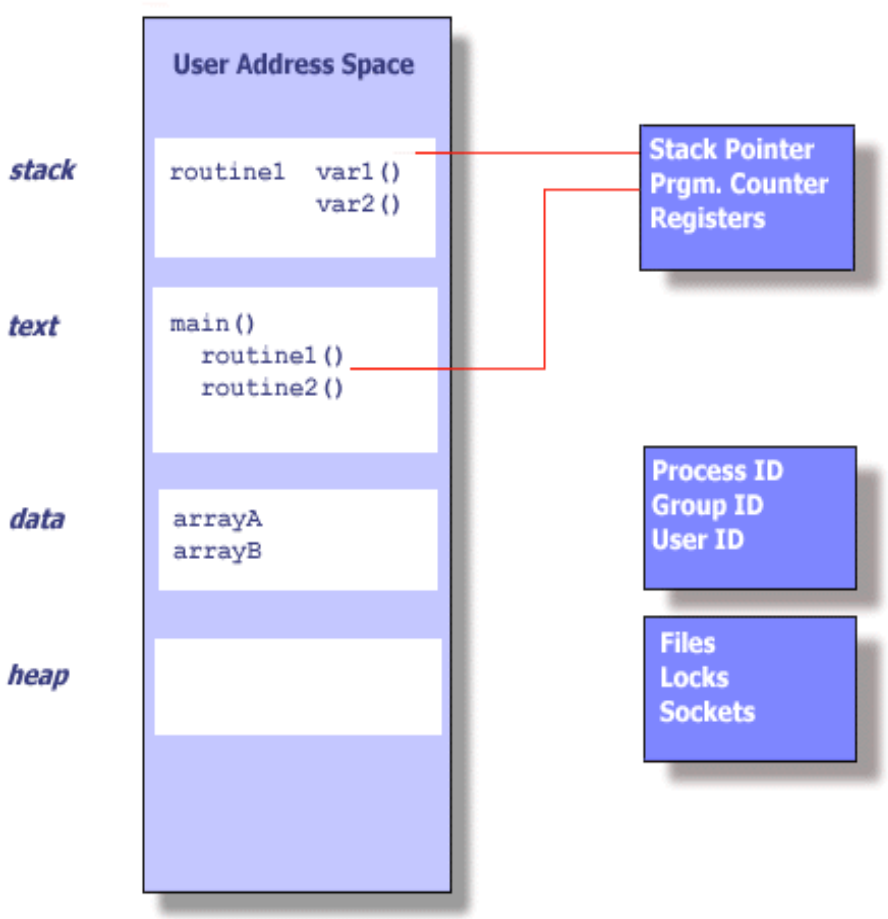
O Descritor de Thread



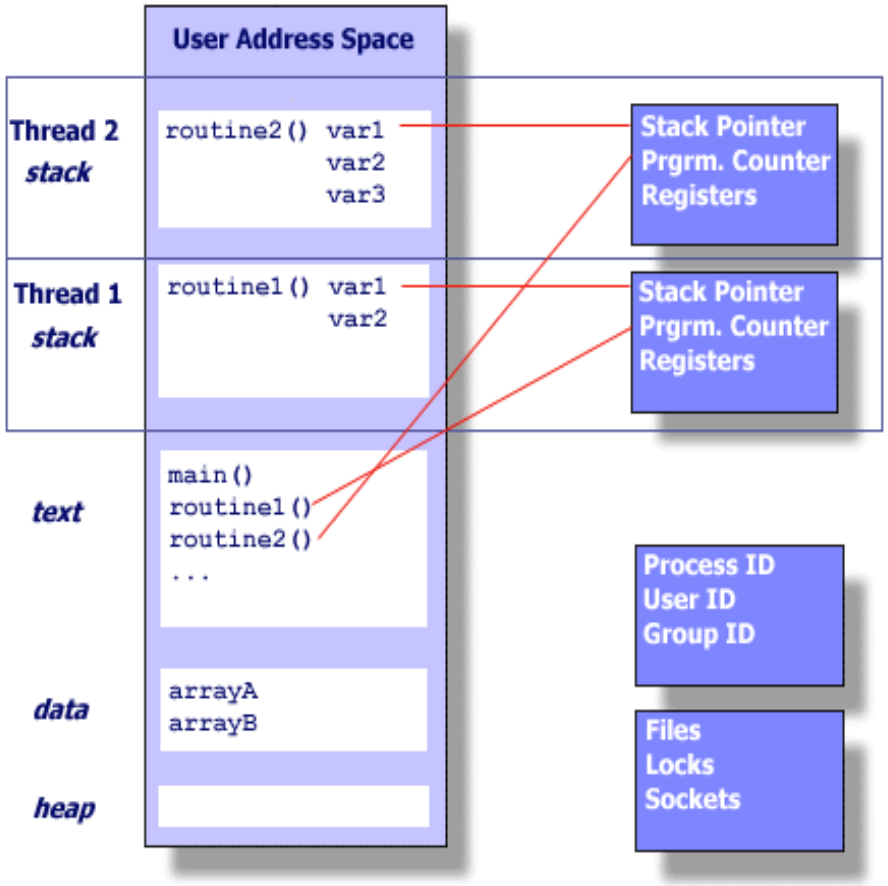
- Para cada thread, o kernel (ou biblioteca de threads) mantém a seguinte informação, que é mantida independente dos descritores de processos (PCBs)

```
Contexto:
    program counter (PC)          /* próxima instrução */
    process status word (PSW)     /* resultado da  operação, ex: carry-bit */
    stack pointer (SP)            /* pilha de execução do thread */
    registers                     /* conteúdo dos registradores da CPU */
    state                         /* blocked, running, ready */
    priority
    host_process                  /* processo hospedeiro ou kernel */
    thread_Id                     /* identificador do thread */
    processID                     /* processo ao qual a thread pertence */
```

Gerenciamento processos vs threads



UNIX PROCESS

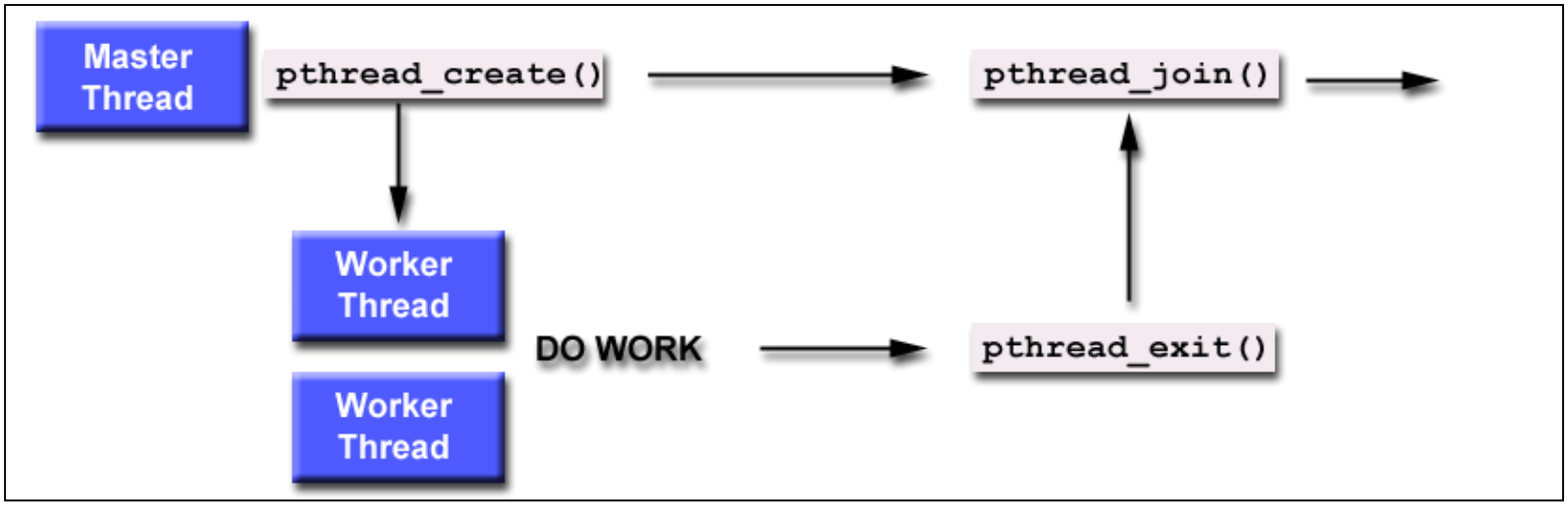


THREADS WITHIN A UNIX PROCESS

Sincronismo entre Threads

int pthread_join(pthread_t tid, void* status)
// a thread invocadora é bloqueada até que a thread tid termine

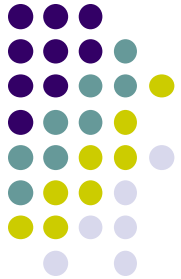
- tid A threadID pela qual deseja-se esperar;
- status O valor de retorno da thread execurando o exit(), será copiado para status




```
void main() {
        pthread_t tid;
        int status;
              pthread_create(&tid,NULL,thread_main,NULL);
        ….
        pthread_join(tid, (void*) &status);
        printf("Return value is: %d\n", status);
}
```

```
void *thread_main( ){
        int result;
        ....
        Pthread_exit((void*) result);
}
```

Exemplo de Uso de Threads



```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
        printf("\n%d: Hello World!\n", threadid);
        /* do other things */
        pthread_exit(NULL);             /*not necessary*/
}

int main()
{
        pthread_t threads[NUM_THREADS];
        int t;
        for(t=0;t < NUM_THREADS;t++)
        {
          printf("Creating thread %d\n", t);
          pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        }

        for(t=0; t < NUM_THREADS; t++)
          pthread_join(threads[t],NULL);        /* wait for all the threads to
                                                   terminate*/
}
```

Perguntas?