UNIVERSIDADE ESTADUAL DE PONTA GROSSA

SETOR DE CIÊNCIAS AGRÁRIAS E DE TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

ENGENHARIA DE COMPUTAÇÃO

MATEUS FELIPE DA SILVA JUNGES

SISTEMAS OPERACIONAIS – ATIVIDADES

PONTA GROSSA

2018

1. **O que é escalonamento?**

   R: Quando o sistema operacional é responsável por decidir em que momento cada processo recebe a CPU

2. **Qual mecanismo de escalonamento é implementado no simulador.c?**
   R: O primeiro a chegar é o primeiro a ser servido

3. **Alterar o código e fazer um mecanismo Round Robin completo.**
   R:

4. **Analisar o código mkern0.c. Escalonamento em ambiente monoprogramado (DOS).**
   R:

5. **Material sobre processos, threads e mecanismos de IPC:**

   S02_AP_02.3_01.03.2018__Advanced_Linux_Programming-ch03-processes.pdf
   S02_AP_02.4_01.03.2018__Advanced_Linux_Programming-ch04-threads.pdf
   S02_AP_02.5_01.03.2018__Advanced_Linux_Programming-ch05-ipc.pdf

7. **Para que serve a biblioteca pthread()?**
   R: Biblioteca usada para trabalhar com threads

8. **Listar funções/procedimentos/estruturas de dados referentes a biblioteca de programação Thread Posix.**
   R:
   pthread_create(): Cria uma nova thread no espaço de endereço de onde foi chamada;
   pthread_exit(): Termina o thread chamador;
   pthread_join: Espera pelo término de uma thread;
   pthread_mutex_init: Cria um novo mutex;
   pthread_mutex-destroy: Destrói um mutex;
   pthread_mutex_lock: Impede um mutex;
   pthread_mutex_unlock: Libera um mutex;
   pthread_cond-init: Cria uma variável de condição;
   pthread_cond_destroy: Destrói uma variável de condição;
   pthread_cond_wait: Espera por uma variável de condição;
   pthread_cond_signal: Libera um thread que está à espera de uma variável de condição.

pthread_attr_destroy(pthread_attr_t *):

Quando um objeto de atributos não é mais necessário, ele deve ser destruído usando a função pthread_attr_destroy(). A destruição de um objeto de atributos de encadeamentos não afeta os encadeamentos usando esse objeto.

pthread_attr_getdetachstate(const pthread_attr_t *, int *):

Define o atributo de estado de desanexação do objeto de Atributos de encadeamento referido por attr ao valor especificado em detachstate. O atributo do estado de desanexação determina se um encadeamento criado usando o atributo attr do objeto de encadeamento será criado em um estado que pode ser único ou desanexado.

pthread_attr_getguardsize(const pthread_attr_t *, size_t *):

Define o atributo de tamanho de guarda do objeto de atrivutos de encadeamento referido por attr ao valor especificado em guarsize. Se guardsize for maior que 0, então para cada novo thread criado usando attr o sistema aloca uma região adicional de pelo menos guardsize no final da pilha do thread para atuar como área de guarda para a pilha. Se guardsize for igual a 0, os novos threads criados com attr não terão uma área de guarda. O tamanho de guarda padrão é o mesmo tamanho da página do sistema.

pthread_attr_getinheritsched(const pthread_attr_t *, int *):

Define a herança atributo scheduler do objeto de atributos de encadeamento referido por attr para o valor especificado em inheritsched . O agendador de herança atributo determina se um encadeamento criado usando o encadeamento atributos objeto attr herdará seus atributos de agendamento de o segmento de chamada ou se ele vai levá-los de attr .

pthread_attr_getschedparam(const pthread_attr_t *,
struct sched_param *):

Define os atributos do parâmetro de escalonamento do objeto de atributos de encadeamento referido por *attr* para os valores especificados no buffer apontado por *param* . Esses atributos determinam os parâmetros de planejamento de um encadeamento criado usando os atributos de encadeamento objeto *attr* .

pthread_attr_getschedpolicy(const pthread_attr_t *, int *):

Define o atributo de política de planejamento do objeto de atributos de encadeamento referido por *attr* para o valor especificado na *política* . Esse atributo determina a política de planejamento de um encadeamento criado usando os atributos de encadeamento objeto *attr* .

pthread_attr_getscope(const pthread_attr_t *, int *):

define o atributo de escopo de contenção do objeto de atributos de encadeamento referido por *attr* ao valor especificado no *escopo* . O atributo escopo de contenção define o conjunto de encadeamentos com o qual um encadeamento compete por recursos, como a CPU. POSIX.1-2001 especifica dois valores possíveis para o *escopo* :

PTHREAD_SCOPE_SYSTEM
O encadeamento compete por recursos com todos os outros encadeamentos em todos os processos no sistema que estão no mesmo domínio de alocação de planejamento (um grupo de um ou mais processadores). **Os** encadeamentos **PTHREAD_SCOPE_ SYSTEM** são agendados em relação um ao outro de acordo com sua política e prioridade de agendamento.

PTHREAD_SCOPE_PROCESS
O encadeamento compete por recursos com todos os outros encadeamentos no mesmo processo que também foram criados com o escopo de contenção **PTHREAD_SCOPE_PROCESS** . **Os**encadeam entos **PTHREAD_SCOPE_PROCESS** são planejados em relação a outros encadeamentos no processo, de acordo com sua política de agendamento e prioridade. POSIX.1-2001 não especifica como esses threads lidam com outros threads em outro processo no sistema ou com outros threads no mesmo processo que foram criados com o escopo de contenção **PTHREAD_SCOPE_SYSTEM** .

POSIX.1-2001 requer apenas que uma implementação suporte um desses escopos de contenção, mas permite que ambos sejam suportados. O Linux suporta **PTHREAD_SCOPE_SYSTEM** , mas não **PTHREAD_SCOPE_PROCESS** .

A função **pthread_attr_getscope** () retorna o atributo de escopo de contenção do objeto de atributos de encadeamento referido por *attr* no buffer apontado por *escopo* .

pthread_attr_getstackaddr(const pthread_attr_t *, void **);

A função **pthread_attr_setstackaddr** () define o atributo de endereço de pilha do objeto de atributos de encadeamento referido por *attr* ao valor especificado em *stackaddr* . Este atributo especifica a localização da pilha que deve ser usada por um encadeamento criado usando o atributo *attr do* objeto de encadeamento .

O stackaddr deve apontar para um buffer de pelo menos **PTHREAD_STACK_MIN** bytes alocados pelo chamador. As páginas do buffer alocado devem ser legíveis e graváveis.

A função **pthread_attr_getstackaddr** () retorna o atributo de endereço de pilha do objeto de atributos de encadeamento referido por *attr* no buffer apontado por *stackaddr* .

pthread_attr_getstacksize(const pthread_attr_t *, size_t *);

A função **pthread_attr_setstacksize** () define o atributo de tamanho de pilha do objeto de atributos de encadeamento referido por *attr* para o valor especificado em *tamanho de encadernação* .

O atributo de tamanho de pilha determina o tamanho mínimo (em bytes) que será alocado para encadeamentos criados usando os atributos de encadeamento objeto *attr* .

A função **pthread_attr_getstacksize** () retorna o atributo de tamanho de pilha do objeto de atributos de encadeamento referido por *attr* no buffer apontado por *stacksize* .

pthread_attr_init(pthread_attr_t *):

inicializa o objeto de atributos de encadeamento apontado por *attr* com valores de atributo padrão. Após essa chamada, os atributos individuais do objeto podem ser definidos usando várias funções relacionadas e, em seguida, o objeto pode ser usado em uma ou mais chamadas **pthread_create** que criam threads.

Chamar **pthread_attr_init** () em um objeto de atributos de encadeamento que já foi inicializado resulta em um comportamento indefinido.

pthread_attr_setdetachstate(pthread_attr_t *, int):

Define o atributo de estado de desanexação do objeto de atributos de encadeamento referido por *attr* ao valor especificado em *detachstate* . O atributo do estado de desanexação determina se um encadeamento criado usando o atributo *attr do* objeto de encadeamento será criado em um estado que pode ser unido ou desanexado.

pthread_attr_setguardsize(pthread_attr_t *, size_t):

define o atributo de tamanho de guarda do objeto de atributos de encadeamento referido por *attr* ao valor especificado em *guardsize* . Se *guardsize* for maior que 0, então para cada novo thread criado usando *attr* o sistema aloca uma região

adicional de pelo menos *guardsize* no final da pilha do thread para atuar como a área de guarda para a pilha (mas veja BUGS). Se *guardsize* for 0, os novos threads criados com *attr* não terão uma área de guarda. O tamanho de guarda padrão é o mesmo que o tamanho da página do sistema.

pthread_equal:

Compara duas trheads.

pthread_exit:

termina o thread de chamada e retorna um valor via *retval* que (se o thread for joinable) está disponível para outro thread no mesmo processo que chama ***pthread_join*** *(3)* .

pthread_join:

aguarda o término do encadeamento especificado pelo *encadeamento* . Se esse segmento já tiver terminado, **pthread_join** () retornará imediatamente. O encadeamento especificado pelo *encadeamento* deve ser joinable.

pthread_mutex_destroy:

deve destruir o objeto mutex referenciado por *mutex* ; o objeto mutex torna-se, na verdade, não inicializado. Uma implementação pode causar *pthread_mutex_destroy* () para definir o objeto referenciado por *mutex* para um valor inválido. Um objeto mutex destruído pode ser reinicializado usando *pthread_mutex_init* (); os resultados de referenciar o objeto depois de ter sido destruído são indefinidos.

pthread_mutex_init:

deve inicializar o mutex referenciado por *mutex* com atributos especificados por *attr* . Se *attr* é NULL, os atributos mutex padrão são usados; o efeito deve ser o mesmo que passar o endereço de um objeto de atributo mutex padrão. Após a inicialização bem-sucedida, o estado do mutex torna-se inicializado e desbloqueado.

pthread_mutex_lock:

O objeto mutex referenciado por *mutex* deve ser bloqueado chamando *pthread_mutex_lock* (). Se o mutex já está bloqueado, o segmento de chamada deve bloquear até que o mutex fique disponível. Esta operação deve retornar com o objeto mutex referenciado por *mutex* no estado bloqueado com o segmento de chamada como seu proprietário.

pthread_mutex_setprioceiling:

deve retornar o teto de prioridade atual do mutex.

pthread_mutex_unlock:

O objeto mutex referenciado por *mutex* deve ser bloqueado chamando *pthread_mutex_lock* (). Se o mutex já está bloqueado, o segmento de chamada deve bloquear até que o mutex fique disponível. Esta operação deve retornar com o objeto mutex referenciado por *mutex* no estado bloqueado com o segmento de chamada como seu proprietário.

pthread_mutexattr_destroy:

deve destruir um objeto de atributos mutex; o objeto se torna, na verdade, não inicializado. Uma implementação pode causar *pthread_mutexattr_destroy* () para definir o objeto referenciado por *attr* para um valor inválido. Um objeto de atributo *attr* destruído pode ser reinicializado usando *pthread_mutexattr_init* (); os resultados de referenciar o objeto depois de ter sido destruído são indefinidos.

pthread_mutexattr_setpshared:

deve obter o valor do atributo *compartilhado* pelo *processo* do objeto de atributos referenciado por *attr* . A função *pthread_mutexattr_setpshared* () deve definir o atributo *compartilhado* pelo *processo* em um objeto de atributos inicializados referenciado por *attr* .

pthread_mutexattr_settype:

Deve obter e definir o atributo do *tipo* mutex . Este atributo é definido no parâmetro *type* para essas funções. O valor padrão do atributo *type* é PTHREAD_MUTEX_DEFAULT.

pthread_once:

por qualquer thread em um processo, com um determinado *once_control* , deve chamar o *init_routine* sem argumentos. Chamadas subseqüentes de *pthread_once* () com o mesmo *once_control* não devem chamar o *init_routine* . No retorno de *pthread_once* (), o*init_routine* deve ser concluído. O parâmetro *once_control* deve determinar se a rotina de inicialização associada foi chamada.

pthread_rwlock_destroy:

Deve destruir o objeto de bloqueio de leitura-gravação referenciado pelo *rwlock* e liberar quaisquer recursos usados pelo bloqueio. O efeito do uso subseqüente do bloqueio é indefinido até que o bloqueio seja reinicializado por outra chamada para *pthread_rwlock_init* (). Uma implementação pode

causar *pthread_rwlock_destroy* () para definir o objeto referenciado pelo *rwlock* para um valor inválido. Os resultados são indefinidos se *pthread_rwlock_destroy* () for chamado quando qualquer thread contiver *rwlock* . A tentativa de destruir um bloqueio de leitura / gravação não inicializado resulta em comportamento indefinido.

pthread_rwlock_rdlock:
deve aplicar um bloqueio de leitura ao bloqueio de leitura / gravação referenciado pelo *rwlock* . O encadeamento de chamada adquire o bloqueio de leitura se um gravador não mantiver o bloqueio e não houver gravadores bloqueados no bloqueio.

pthread_rwlock_unlock:
deve liberar um bloqueio mantido no objeto de bloqueio de leitura-gravação referenciado pelo *rwlock* . Os resultados são indefinido se o bloqueio de leitura e gravação *rwlock* não é mantido pelo thread de chamada.

pthread_rwlock_wrlock:
Deve aplicar um bloqueio de gravação como
a função *pthread_rwlock_wrlock* (), com a exceção de que a função falhará se qualquer thread atualmente contiver *rwlock* (para leitura ou escrita).

pthread_rwlockattr_destroy:
deve destruir um objeto de atributos de bloqueio de leitura / gravação. Um objeto de atributo *attr* destruído pode ser reinicializado usando *pthread_rwlockattr_init* ();

pthread_rwlockattr_getpshared:
Deve obter o valor do atributo *compartilhado* pelo *processo* do objeto de atributos inicializados referenciado por *attr*

pthread_*rwlockattr*_init:
Deve destruir um objeto de atributos de bloqueio de leitura / gravação. Um objeto de atributo *attr* destruído pode ser reinicializado usando *pthread_rwlockattr_init* (); os resultados de referenciar o objeto depois de ter sido destruído são indefinidos. Uma implementação pode causar *pthread_rwlockattr_destroy* () para definir o objeto referenciado por *attr* para um valor inválido.

pthread_rwlockattr_setpshared:
Deve obter o valor do atributo *compartilhado* pelo *processo* do objeto de atributos inicializados referenciado por *att*

pthread_self:

retorna o ID do segmento de chamada

pthread_setcanceltype:

O **pthread_setcancelstate** () define o estado de cancelabilidade do encadeamento de chamada para o valor dado no *estado* . O estado de cancelabilidade anterior do encadeamento é retornado no buffer apontado por *oldstate* . O argumento de *estado* deve ter um dos seguintes valores:

PTHREAD_CANCEL_ENABLE

O encadeamento é cancelável. Esse é o estado de cancelabilidade padrão em todos os novos threads, incluindo o thread inicial. O tipo de cancelabilidade do encadeamento determina quando um encadeamento cancelável responderá a um pedido de cancelamento.

PTHREAD_CANCEL_DISABLE

O encadeamento não é cancelável. Se uma solicitação de cancelamento for recebida, ela será bloqueada até que a capacidade de cancelamento seja ativada.

O **pthread_setcanceltype** () define o tipo de cancelabilidade do encadeamento de chamada para o valor dado em *type* . O tipo de cancelabilidade anterior do encadeamento é retornado no buffer apontado por *oldtype* . O argumento *type* deve ter um dos seguintes valores:

PTHREAD_CANCEL_DEFERRED

Uma solicitação de cancelamento é adiada até que o thread em seguida chame uma função que é um ponto de cancelamento (consulte ***pthreads*** *(7)*). Esse é o tipo de cancelabilidade padrão em todos os novos encadeamentos, incluindo o encadeamento inicial.

PTHREAD_CANCEL_ASYNCHRONOUS

O segmento pode ser cancelado a qualquer momento. (Normalmente, ele será cancelado imediatamente após receber uma solicitação de cancelamento, mas o sistema não garante isso.)

pthread_setconcurrency:

Informa a implementação do nível de simultaneidade desejado do aplicativo, especificado em *new_level* . A implementação só leva isso como uma dica: POSIX.1 não especifica o nível de simultaneidade que deve ser fornecido como resultado da chamada **pthread_setconcurrency** ().

pthread_setschedparam:

Define a política de agendamento e os parâmetros do encadeamento de *encadeamento* .

pthread_setspecific:

Deve retornar o valor atualmente vinculado à *chave* especificada em nome do segmento de chamada.

pthread_testcancel:

cria um ponto de cancelamento dentro do encadeamento de chamada, para que um encadeamento que, de outra forma, esteja executando código que não contenha pontos de cancelamento, responda a uma solicitação de cancelamento.

Se a cancelabilidade estiver desabilitada (usando ***pthread_setcancelstate*** *(3)*), ou nenhuma solicitação de cancelamento estiver pendente, uma chamada para **pthread_testcancel** () não **terá** efeito.

9. Em relação ao simulador de escalonamento, descreva com detalhes a estrutura sigaction. Relacione os sinais (signals) que existem.

R: [ARQUIVO "simulador.c – comentado.pdf"]

10. Pesquise sobre as seções do manual, descreva quais são e como encontrar texto em vários.

R: O manual possui várias seções:

(1) Programas .exe ou comandos Shell
(2) Chamadas de sistema (funções fornecidas pelo kernel)
(3) Chamadas de bibliotecas (funções de bibliotecas)
(4) Arquivos especiais
(5) Formatos e convenções de arquivos
(6) Jogos
(7) Diversos. Inclui pacotes de macro e convenções
(8) Comandos de administração do sistema
(9) Rotinas do kernel.

Para encontrar textos no manual, é possível digitar '/' e o texto procurado.