

Função Clone: Chamada de sistema usada para criar processos chamados de threads ou tarefas.

- Ela permite uma grande flexibilidade na forma como o novo processo compartilha recursos com o processo chamador (o processo pai).
- Ao chamar clone, você pode especificar um conjunto de sinalizadores que determinam exatamente quais recursos são compartilhados entre o processo pai e o filho. Nesse caso consistem em clone_fs, clone_files, clone_sighand, clone_vm
- A diferença para um fork() é que nessa função observamos um alto grau de controle sobre o que é compartilhado entre o processo pai e filho.

CLONE_FS

- O processo filho compartilha as mesmas informações do sistema de arquivos que o processo pai, isto é, diretório de trabalho atual e raiz do sistema de arquivos.

CLONE_FILES

- O processo filho compartilha as mesmas informações do sistema de arquivos que o processo pai, isto é, os mesmos descritores/diretórios de arquivos abertos.

CLONE_SIGHAND

- O processo filho e pai compartilham a mesma tabela de manipuladores de sinal.

CLONE_VM

- O processo filho e pai compartilham o mesmo espaço de endereçamento de memória, permitindo uma comunicação muito direta entre cada thread, visto que ambos podem acessar e modificar as mesmas variáveis e estruturas de dados da memória.

Resumo do código

O processo pai aloca uma pilha de memória para o novo processo e o cria usando clone(). O processo filho executará a função threadFunction, enquanto o pai continua a execução logo após a chamada de clone().

O processo pai modifica a variável var_compartilhada para 1, sinalizando ao processo filho. O filho está em um loop de espera ativa (usleep), esperando até que var_compartilhada mude de 0 para 1. Ao detectar essa mudança, imprime uma mensagem de recebimento e altera var_compartilhada para 2.

O processo pai, que estava em um loop de espera ativa (usleep), esperando que seu filho mude seu valor para 2, recebe a confirmação do filho, e dessa forma, o processo pai espera pelo término do processo filho usando waitpid(), e libera os recursos alocados do processo filho, termina assim sua execução.

OBS:

1) Variável Volátil (var_compartilhada): Como é criado um processo que compartilha o mesmo espaço de memória, é permitido a comunicação direta através da modificação de variáveis globais, ou seja, consiste em “dizer” ao compilador não assumir nada sobre o estado dessa variável além do que for explicitamente manipulado pelo programa.

```
pid = clone(&threadFunction, (char*)stack + FIBER_STACK, SIGCHLD | CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM, 0);
```

Nesta linha do código, é onde cria-se um novo processo, ou seja, as threads, em que os parâmetros passados consistem em:

- &threadFunction → a função filho que vai ser executada pelo processo, passada anteriormente;
- (char*)stack + FIBER_STACK → O ponteiro aponta para o topo da pilha da memória alocada para o novo processo;
- SIGCHLD | CLONE_FS | CLONE_FILES | CLONE_SIGHAND | CLONE_VM → São passados os flags que especificam o comportamento do clone();
- 0 → É utilizado para passar dados.

Execução do código no editor de texto (nano) dentro do servidor da AWS:

The screenshot displays the AWS Academy lab interface. At the top, the breadcrumb navigation shows 'ALLv1PT-...' > 'Módulos' > 'Laboratóri...'. Below this, a link says '> Iniciar os laboratórios de aprendizagem da AWS Academy'. The lab status bar indicates 'AWS' is active, 'Used \$0.6 of \$100', and the time is '03:52'. On the left sidebar, there are links for 'Página inicial', 'Módulos', and 'Fóruns'. The main area features a terminal window titled 'lab03.c' running 'GNU nano 5.6.1'. The code in the terminal is as follows:

```
    }
    usleep(100);
    printf("Filho: Recebi a mensagem!\n");

    var_compartilhada = 2;

    return 0;
}

int main() {
    void* stack;
    pid_t pid;

    stack = malloc(FIBER_STACK);
    if (stack == 0) {
        perror("malloc: Could not allocate stack");
        exit(1);
    }

    printf("Thread filho eh criado\n");
```

Below the code, a row of keyboard shortcuts is visible: Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Execute, Justify, Location, Go To Line, M-E, Undo, M-E, Redo. On the right side of the terminal, there's a dropdown menu set to 'PT-BR' and a section titled 'Laboratório de aprendizagem m' with links: 'Visão geral do ambiente', 'Navegação pelo ambiente', 'Acessar o console de gerenciamento da AWS', and 'Reset'.

At the bottom of the interface, there are navigation buttons: '◀ Anterior' and 'Próximo ▶'.

Execução do código com gcc dentro do servidor da AWS:

ALLy1PT-... > Módulos > Laboratóri...

▶ Iniciar os laboratórios de aprendizagem da AWS Academy

AWS Used \$0.6 of \$100 03:50 ▶ Start Lab ■ End Lab ⓘ AWS Details ⓘ Readme ↺ Reset ✕

Página inicial

Módulos

Fóruns

Verifying : pkgconf-m4-1.7.3-10.e19.noarch 11/12
Verifying : pkgconf-pkg-config-1.7.3-10.e19.x86_64 12/12
Installed products updated.

Installed:
cpp-11.4.1-2.1.e19.x86_64 gcc-11.4.1-2.1.e19.x86_64
glibc-devel-2.34-83.e19_3.7.x86_64 glibc-headers-2.34-83.e19_3.7.x86_64
kernel-headers-5.14.0-362.24.1.e19_3.x86_64 libmpc-1.2.1-4.e19.x86_64
libpkgconf-1.7.3-10.e19.x86_64 libxcrypt-devel-4.4.18-3.e19.x86_64
make-1:4.3-7.e19.x86_64 pkgconf-1.7.3-10.e19.x86_64
pkgconf-m4-1.7.3-10.e19.noarch pkgconf-pkg-config-1.7.3-10.e19.x86_64

Complete!
[ec2-user@ip-172-31-18-251 lab03]\$ ls
lab03.c
[ec2-user@ip-172-31-18-251 lab03]\$ gcc lab03.c
[ec2-user@ip-172-31-18-251 lab03]\$ ls
a.out lab03.c
[ec2-user@ip-172-31-18-251 lab03]\$./a.out
Thread filho eh criado
Pai: Enviei a mensagem!
Filho: Recebi a mensagem!
Pai: Filho confirmou o recebimento da mensagem.
Thread filho retornado e pilha liberada.
[ec2-user@ip-172-31-18-251 lab03]\$ []

PT-BR

Laboratório de aprendizagem

[Visão geral do ambiente](#)
[Navegação pelo ambiente](#)
[Acessar o console de gerenciamento da AWS](#)

◀ Anterior

Próximo ▶

Referências:

<https://eli.thegreenplace.net/2018/launching-linux-threads-and-processes-with-clone/>