Trabalho 1 - Inicialização de programas e comunicação entre processos

DEL/Poli/UFRJ

2021/1

1 Entrega e Pontuação

A data de entrega dos trabalhos, bem como sua pontuação, está especificada no arquivo das regras da disciplina. **Não haverá adiamento da entrega**. O código deve ser entregue em um arquivo .zip pelo e-mail **rodsouzacouto@poli.ufrj.br**. O assunto do e-mail enviado deverá iniciar com o prefixo [EEL770][T1]. **ATENÇÃO: E-mails com assunto que não se iniciem com o prefixo serão ignorados. Não use esse prefixo para tirar dúvidas sobre o trabalho.**

Coloque seu nome e sobrenome no arquivo .zip, sem caracteres de espaço. Ao zipar um diretório, certifique-se que tal diretório possui seu nome e sobrenome. Ou seja, **não** zipe um diretório com um nome genérico (p.ex., "Trabalho 1") para depois alterar o nome do arquivo zip. **ATENÇÃO: Não envie executáveis, pois serão rejeitados pelo gmail. Enviem apenas o código fonte, com o Makefile.**

2 Objetivo do Trabalho

Neste trabalho você fará um programa que chamaremos de disparador e executa duas tarefas diferentes. O início da execução das tarefas será disparado por uma recepção de sinais. Siga exatamente as exigências do trabalho.

O trabalho não representa nenhuma ferramenta usada comercialmente. Ele apenas objetiva o conhecimento prático das principais funções usadas no gerenciamento de processos. Por exemplo, todas as funções vistas neste trabalho são usadas por shells. Assim, algumas tarefas podem não fazer sentido do ponto de vista prático, mas servem para explicitar alguns comportamentos das funções utilizadas.

Este trabalho assume que você leu o Capítulo 5 do livro (`https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf`) e viu o vídeo da aula de Comunicação entre processos com pipes.

3 Funcionamento básico

Ao iniciar a execução, o programa imprimirá na tela seu pid e inicializará com o valor zero uma variável denominada `comandoParaExecutar`. Após isso, o disparador ficará esperando sinais recebidos externamente. Os sinais recebidos podem ser SIGUSR1, SIGUSR2 ou SIGTERM. Se o sinal recebido for SIGUSR1, o disparador deverá executar a **Tarefa 1** descrita a seguir. Se o sinal recebido for SIGUSR2, o disparador deverá executar a **Tarefa 2** descrita a seguir. Se o sinal recebido for SIGTERM, o programa deverá imprimir na tela "Finalizando o disparador..." e o disparador deverá finalizar sua execução (ou seja, return 0 da função main).

Após a execução de cada tarefa, o programa deverá voltar a esperar sinais. "Esperar sinais" significa ficar em um loop infinito, visto que os sinais são tratados pelos *handlers*. Veja detalhes no material mencionado na Seção 4.

3.1 Tarefa 1

A Tarefa 1 consiste nos seguintes passos:

1. O pai cria um pipe;
2. O pai cria um processo filho;
3. O pai fica esperando a finalização do filho;
4. O filho sorteia um número inteiro aleatório de 1 a 100;
5. O filho imprime o número na tela e envia esse número para o pai. Esse envio deverá ser realizado via pipe.
6. O filho fecha qualquer ponta aberta do pipe e finaliza;
7. O pai lê o valor recebido e guarda a resposta do filho na variável denominada `comandoParaExecutar`;
8. O pai fecha qualquer ponta aberta do pipe;

3.2 Tarefa 2

A Tarefa 2 consiste nos seguinte passos:

1. O pai cria um processo filho;
2. O pai envia ao filho o conteúdo da variável `comandoParaExecutar`;
3. O pai fica esperando a finalização do filho;
4. O filho verifica o número recebido. Dependendo do valor do número, o filho executa uma das seguintes ações:
   - Se o número for zero (ou seja, a Tarefa 1 nunca foi chamada) o filho imprime "Não há comando a executar" e finaliza;
   - Se o número for diferente de zero e par, o filho executa o comando "ping 8.8.8.8 -c 5" por meio de uma chamada exec;
   - Se o número for ímpar, o filho executa o comando "ping paris.testdebit.info -c 5 -i 2" por meio de uma chamada exec.

Não é permitido utilizar pipe na Tarefa 2. O pai consegue "enviar" o conteúdo da variável `comandoParaExecutar` para o filho sem precisar de pipe. Lembre-se da cópia do pai realizada pela chamada fork.

4 Recepção de sinais

O disparador deve estar preparado para receber os sinais SIGUSR1, SIGUSR2 ou SIGTERM.

O envio de sinais se dará por meio de um shell em outro terminal como, por exemplo, com o comando "kill -SIGUSR1 pidDoProcesso" .

Veja algumas informações práticas sobre sinais em
`https://www.thegeekstuff.com/2012/03/linux-signals-fundamentals/` e
`https://www.thegeekstuff.com/2012/03/catch-signals-sample-c-code/`. Fiz um resumo sobre o tema em `https://drive.google.com/file/d/1EieO-jz3AGPHuaQgZclyYWnjVl8fLV-0/view?usp=sharing`. Lembrando que é necessário logar com o PoliMail ou ter previamente pedido autorização, para quem não é da Poli.

Dica: Para saber o pid de um processo, faça o comando "ps aux | grep nomeDoProcesso".

5 Outras Exigências

- O programa deve ser escrito em C ou C++;
- O código poderá ser desenvolvido para o Linux ou para o MacOS. Informe no início do código qual sistema operacional você usou para fazer o trabalho. No caso de uma distribuição Linux, informe qual é a distribuição (Ubuntu, Debian, VM fornecida na disciplina, etc.);
- Não é permitido usar popen() ou outras funções que abram shells. A criação do processo filho deverá ser realizada por um fork e a execução de comandos (Tarefa 2) deverá ser realizada a partir da chamada exec;
- Como comunicação entre processos, não use sockets ou FIFOs para não complicar à toa o trabalho. Use pipes;
- O código deve vir acompanhado de Makefile. Códigos que não forem entregues em condições de serem compilados serão zerados. O seguinte código tem exemplo de MakeFile `https://drive.google.com/file/d/1wWEsfPYXpFlcPJZywxCdjW_JX4oEA-aJ/view?usp=sharing`. Antes de entregar o trabalho, testem se o código compila com o MakeFile.;
- Lembrem-se de escolher nomes adequados para as variáveis e identar;
- Muito importante! O código deve ser comentado para permitir o entendimento do raciocínio utilizado por vocês ao escrever o programa;
- O trabalho deve ser realizado de forma individual;
- É aceitável que você use pequenos trechos de código da Internet (p.ex., stackoverflow). Caso use algum trecho da Internet, faça um comentário antes do trecho dizendo de onde o código foi retirado. Caso não haja essa referência, considerarei que você copiou o trabalho da Internet, sendo considerado cópia de trabalho;
- Não use código de colegas que já cursaram Sistemas Operacionais. Esse tipo de cópia é facilmente detectável pelo script de verificação de plágio.
- **Atentem-se para as regras de cópia de trabalho desta disciplina. Não haverá exceções!!!!!**

6 Dicas

Dica1: Para quem quiser aprender a utilizar a chamada fork e exec, o seguinte código pode ser útil `https://drive.google.com/file/d/1wWEsfPYXpFlcPJZywxCdjW_JX4oEA-aJ/view?usp=sharing`.

Dica 2: Para gerar número aleatórios, veja as informações no seguinte site `https://www.tutorialspoint.com/c_standard_library/c_function_srand.htm`. Use o tempo como argumento do srand, como descrito no link anterior, para evitar que a Tarefa 1 sempre sorteie o mesmo número.

Dica 3: As funções read e write leem e escrevem em ponteiros. O exemplo da aula de pipes não mostra ponteiro para escrita pois era um string constante. Entretanto, em qualquer outro caso, o write escreve no pipe o conteúdo do ponteiro indicado no seu argumento. Veja as entradas do man `https://linux.die.net/man/3/read` e `https://linux.die.net/man/2/write`.

Dica4: O seguinte link possui uma VM com Linux para uso no Virtualbox. Seu uso não é obrigatório, mas pode ajudar quem não tiver muita memória RAM para máquina virtual: `https://drive.google.com/file/d/1-R9i3QB3aqnLMBQhUlzO3K2lM8yvxp7P/view?usp=sharing`. A senha da VM, e o nome do usuário, é eel770.