Leia atentamente as instruções pois relatórios fora do formato não serão aceitos.

Instruções

O relatório deve ser enviado no formato PDF.

Toda vez que for solicitado um *print* da tela, você deve realçar (destacar com alguma cor diferente), na imagem, o conteúdo que se deseja mostrar. Além disso, deve-se explicar textualmente o que está sendo destacado. Se isso não for feito, a resposta da questão não será considerada.

Nesta atividade, você não deverá entregar o código fonte. Você deve explicar o raciocínio utilizado para resolver as questões. Você pode colocar um *print* da tela contendo o código fonte.

Teoria para você fazer a atividade

Conforme vimos, o processo pai pode esperar o filho usando a chamada de sistema **pid=wait(&status)**. Esta chamada retorna o status e o pid to processo encerrado. Neste contexto, dois conceitos são importantes:

Processo zombie: existe o processo pai, porém ele não está esperando filho (não invocou o wait). Quando o filho termina e o pai existe, mas não está esperando, então o filho se torna zombie. O processo filho deixa de existir na memória, porém seu registro continua no PCB do SO.

Processo órfão: o processo pai termina sem invocar o wait. Neste caso, algum processo do SO se torna pai do processo.

Atividade

1) Analise o programa **fork0.c** e responda.

*Tanto o processo pai quanto o processo filho poderão executar a chamada de função "**sleep(5)**"?*

---

2) Analise o programa **fork1.c** e responda.

*O comando "**printf("%d terminou\n",id);**" irá imprimir o id do processo pai ou do processo filho?*

---

3) Analise o programa **fork2.c** e responda.

*O comando "**printf("%d terminou com status %d\n",id,WEXITSTATUS(status));**" imprime o retorno de qual processo? Altere o código de tal maneira que seja impressso um valor de retorno diferente.*

---

4) Analise o programa **fork-print.c** e responda.

a) a variável "x" pertence a qual processo?

b) o comando x++ altera o valor da variável x em qual processo?

---

5) Execute o fork-execve1.c e explique por que o *pid* do simple não é alterado com o comando execve.

---

6) No programa fork-execve1.c, inserira um *loop* demorado antes do comando *fork*. Execute o programa e anote o tempo de execução. Em seguida, no mesmo programa, coloque o *loop* demorado depois do comando *fork* (ou seja, retire o *loop* que estava antes do *fork* e coloque-o depois do *fork*). Anote e compare o tempo de execução nos dois casos.

Note que, dentro do simple.c existe um procedimento para imprimir o tempo de execução. Se o simple.c é o mesmo nos dois casos, então por que o tempo é diferente? Se as áreas de código, dados, heap e pilha são as mesmas, como o tempo pode ser diferente?

---

7) Modifique os programas fork-execve2.c e simple.c, de tal forma que os 20 filhos criados pelo fork-execve2 se tornem *zombies* e sejam herdados por algum processo do SO.

7.1) Explique o que foi feito no código para gerar os processos *zombies*. Coloque um *print* da tela contendo a parte do código que foi modificada. Explique o raciocínio seguido para deixar os processos *zombies*.

7.2) Execute o comando $top e mostre, na saída, a quantidade de processos *zombies*. Dê um *print* na tela.

---

8) Modifique os programas fork-execve2.c e simple.c, de tal forma que os 20 filhos criados pelo fork-execve2 se tornem ór*fãos* e sejam herdados por algum processo do SO.

8.1) Explique o que foi feito no código para gerar os processos *órfãos*. Coloque um *print* da tela contendo a parte do código que foi modificada. Explique o raciocínio seguido para deixar os processos *órfãos*.

8.2)

No fork-execve2.c existe o seguinte printf

printf ("Sou %5d, filho de %5d\n", getpid(), getppid()) ;

No simple.c tem o seguinte printf

printf ("Simple.c - sou %5d, filho de %5d\n", getpid(), getppid()) ;

Apresente um *print* da tela mostrando que os processos filhos inicialmente têm um pai e depois (no simple.c) esse pai é alterado.

8.3) Qual o nome do processo que herdou os processos filhos. Apresente um *print* da tela.