Atividade Experimental 2

Programa para calcular o MÁXIMO ou o MÍNIMO dos números de um arquivo

1 Descrição Geral

O objetivo desta atividade é implementar um programa para determinar o maior ou o menor número inteiro de um arquivo fornecido. O programa deverá realizar a tarefa dividindo a lista de números fornecida no arquivo entre vários processos e, ao final, comparando os resultados fornecidos por cada um desses processos para determinar o maior ou menor.

A programa "sort2" deve ser implementado segundo as especificações deste documento. Além disso, essa é uma atividade INDIVIDUAL.

Essa atividade é OPCIONAL! Mas, assim como as demais "Atividades Experimentais", deverão ser realizadas para que o aluno possa utilizar as notas das atividades experimentais no cálculo da média final dos trabalhos.

As implementações entregues serão corrigidas por um CORRETOR AUTOMÁTICO!

NÃO PODEM SER UTILIZADAS OUTRAS BIBLIOTECAS, EXCETO AS BIBLIOTECAS PADRÃO DO "C".

2 Interface de uso do programa

O programa a ser implementado deverá executar na máquina virtual GNU/Linux fornecida no Moodle da disciplina. O programa deverá ser chamado através da linha de comando e deverá seguir a seguinte especificação:

sort2 [opções]

As [opções] podem aparecer em qualquer ordem e são as seguintes:

- "`-l`" seguido de um número inteiro entre 1 e 9. Indica o número de níveis da hierarquia de processos que deve ser usado pelo programa. O valor "1" indica que todo o programa deve ser implementado no processo principal (sem criar outros processos);
- "`-f`", seguido pelo nome do arquivo com a lista de números inteiros onde procurar pelo maior (menor) valor. Esse arquivo será formado por um conjunto de números inteiros (apenas valores entre "0" e "127"), um número por linha, e em formato texto;
- "`-M`", indica que o programa deve fornecer o MAIOR elemento encontrado no vetor;
- "`-m`", indica que o programa deve fornecer o MENOR elemento encontrado no vetor.

Como resultado, o programa deverá colocar na tela:

- O PID de cada processo criado. Essa informação deverá ser colocada na tela assim que o processo for criado;
- O resultado final da operação do programa. Essa ação é de responsabilidade do processo principal.

Os PIDs dos processos devem ser colocados na tela, um por linha, usando o seguinte formato:

PROC <id> LEVEL <level>

Onde <*id*> é o PID do processo e <*level*> é o nível do processo. Considera-se que o nível do processo principal do programa é o nível "1".

O resultado do processamento deverá ser colocado na tela em uma linha separada das demais informações, usando o seguinte formato:

MAIOR <int> ou ***MENOR <int>***

Onde <*int*> é o maior (menor) valor encontrado.

3 Utilização dos processos pelo programa

Se o programa for chamado com a opção "*-l* 1", não será criado nenhum processo filho e o próprio programa principal deverá ler o arquivo que contém a lista de números, determinar o maior (menor) valor, e colocar as informações solicitadas na tela.

Caso tenha sido solicitado que a implementação utilize mais de um nível de processos, o processo principal deverá implementar sua tarefa através da divisão da lista original contida no arquivo passado como parâmetro na linha de comando em duas partes iguais. Então, deverá distribuir cada uma delas para um processo filho diferente (utilizar a função "*fork*()" para criar os processos filhos).

Essa divisão pela metade e distribuição entre dois processos filhos deverá continuar até chegar ao último nível. Os processos desse último nível devem calcular o maior (menor) valor contido na sublista recebida e informar esse resultado ao seu processo pai. Esse processo pai, por sua vez, determina o maior (menor) valor entre os resultados de seus filhos e repassam o resultado para o seu próprio processo pai.

Esse processo de comparação de dois valores e repasse para o processo pai deve-se repetir até que se chegue ao processo principal, quando o resultado será apresentado na tela.

Na figura 1 está representada a hierarquia de processos para uma chamada do programa com a opção "*-l* 3". Pode-se observar que cada processo tem, sempre, a responsabilidade de criar dois filhos, até que se chegue ao último nível. Todos os processos desse último nível deverão determinar o maior (menor) valor da parte da lista que lhe couber e retornar o resultado para seu pai. Esse, por sua vez, deverá aguardar que os dois filhos terminem sua operação e comparar os dois resultados, retornando para seu processo pai o maior (menor) deles. Esse procedimento se repete até chegar ao processo inicial (principal).

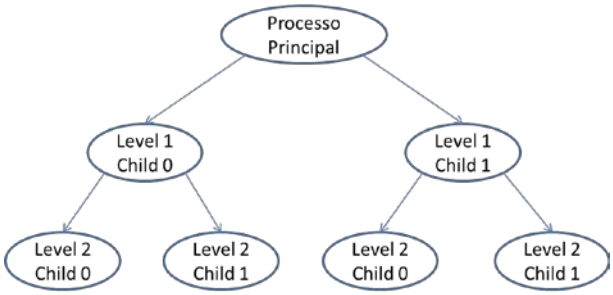

Figura 1 – Exemplo de execução com "*-l* 3"

3.1 Dicas de Operação

Algumas dicas e detalhes do programa:

1. Caso seja selecionada a opção "*-l* 1", não serão criados novos processos e o processo principal terá como responsabilidade realizar toda a tarefa;
2. Ao criar um novo processo, o sistema operacional copia todo o espaço de endereçamento do processo pai para o novo processo filho. Entretanto, quando o processo filho encerrar sua tarefa, seu espaço de endereçamento não retorna para o processo pai. Portanto, para retornar o resultado do processamento de um processo para seu processo pai, deverá ser utilizada a função "*exit*()". Além disso, o processo pai deverá utilizar a função "*waitpid*()" para receber o resultado do processamento de cada um de seus filhos.
3. Para criar processos filhos, um processo pai deve ser utilizada a chamada "***fork*(void)**". Para detalhes de como funciona essa função vide < http://linux.die.net/man/2/fork >.
4. Para retornar o resultado do processamento, um processo filho deve usar uma chamada "***exit*(int status)**". Para detalhes de como funciona essa função, vide <http://linux.die.net/man/3/exit>.
5. Para obter o valor informado na chamada do "*exit*()", um processo deve usar uma chamada "***waitpid*(pid_t pid, int *status, int options)**" para cada um de seus filhos. Para detalhes de como funciona essa função, vide <http://linux.die.net/man/2/wait>.

4 Entregáveis: o que deve ser entregue?

Devem ser entregues:

- Todos os arquivos fonte (arquivos ".c" e ".h") usados para implementar o progrfama;
- Arquivo *makefile* com pelo menos duas regras: "*all*" (para gerar o programa executável, que deve ter o nome de "sort2") e "*clean*" (para apagar todos os arquivos intermediários e o executável.

Os arquivos devem ser entregues em um *tar.gz* (NÃO USAR OUTROS FORMATOS), seguindo, **obrigatoriamente**, a seguinte estrutura de diretórios e arquivos:

\sort2		
	include	DIRETÓRIO: local onde são postos todos os arquivo ".h".
	src	DIRETÓRIO: local onde serão postos todos os arquivos ".c" (códigos fonte) usados na implementação do programa.
	obj	DIRETÓRIO: local onde colocar todos os arquivos do tipo ".o"
	bin	DIRETÓRIO: local onde colocar o arquivo executável, gerado no final do processo de compilação.
	makefile	ARQUIVO: arquivo makefile com as regras para gerar o programa. Deve possuir as regras "*all*" e "*clean*".

5 Avaliação

Para que um trabalho possa ser avaliado ele deverá cumprir com as seguintes condições:

- Entrega dentro dos prazos estabelecidos;
- Obediência à especificação da linha de comando. Se não for obedecida essa determinação, o corretor automático indicará erro na implementação;
- Compilação e geração do executável, sem erros ou *warnings*;
- Fornecimento de todos os arquivos solicitados;

O programa será corrigida de forma automática, e sua valoração será proporcional ao número de casos de teste para os quais programa operar corretamente.

6 Observações

Recomenda-se a troca de ideias entre os alunos. Entretanto, a identificação de cópias de trabalhos acarretará na aplicação do Código Disciplinar Discente e a tomada das medidas cabíveis para essa situação.

ANEXO I – Compilação e Ligação

1. Compilação de arquivo fonte para arquivo objeto

Para compilar um arquivo fonte (*arquivo.c*, por exemplo) e gerar um arquivo objeto (*arquivo.o*, por exemplo), pode-se usar a seguinte linha de comando:

```
gcc –c arquivo.c –Wall
```

Notar que a opção *–Wall* solicita ao compilador que apresente todas as mensagens de alerta (*warnings*) sobre possíveis erros de atribuição de valores a variáveis e incompatibilidade na quantidade ou no tipo de argumentos em chamadas de função.

2. Compilação de arquivo fonte DIRETAMENTE para arquivo executável

A compilação pode ser feita de maneira a gerar, diretamente, o código executável, sem gerar o código objeto correspondente. Para isso, pode-se usar a seguinte linha de comando:

```
gcc –o arquivo arquivo.c –Wall
```

3. Geração de uma biblioteca estática

Para gerar um arquivo de biblioteca estática do tipo "*.a*", os arquivos fonte devem ser compilados, gerando-se arquivos objeto. Então, esses arquivos objeto serão agrupados na biblioteca. Por exemplo, para agrupar os arquivos "*arq1.o*" e "*arq2.o*", obtidos através de compilação, pode-se usar a seguinte linha de comando:

```
ar crs libexemplo.a arq1.o arq2.o
```

Nesse exemplo está sendo gerada uma biblioteca de nome "*exemplo*", que estará no arquivo *libexemplo.a*.

4. Utilização de uma biblioteca

Deseja-se utilizar uma biblioteca estática (chamar funções que compõem essa biblioteca) implementada no arquivo *libexemplo.a*. Essa biblioteca será usada por um programa de nome *myprog.c*.

Se a biblioteca estiver no mesmo diretório do programa, pode-se usar o seguinte comando:

```
gcc -o myprog myprog.c –lexemplo -Wall
```

Notar que, no exemplo, o programa foi compilado e ligado à biblioteca em um único passo, gerando um arquivo executável (arquivo *myprog*). Observar, ainda, que a opção *–l* indica o nome da biblioteca a ser ligada. Observe que o prefixo *lib* e o sufixo *.a* do arquivo não necessitam ser informados. Por isso, a menção apenas ao nome *exemplo*.

Caso a biblioteca esteja em um diretório diferente do programa, deve-se informar o caminho (*path* relativo ou absoluto) da biblioteca. Por exemplo, se a biblioteca está no diretório */user/lib*, caminho absoluto, pode-se usar o seguinte comando:

```
gcc -o myprog myprog.c –L/user/lib –lexemplo -Wall
```

A opção "*-L*" suporta caminhos relativos. Por exemplo, supondo que existam dois diretórios: *testes* e *lib*, que são subdiretórios do mesmo diretório pai. Então, caso a compilação esteja sendo realizada no diretório *testes* e a biblioteca desejada estiver no sudiretório *lib*, pode-se usar a opção *-L* com "*../lib*". Usando o exemplo anterior com essa nova localização das bibliotecas, o comando ficaria da seguinte forma:

```
gcc -o myprog myprog.c –L../lib –lexemplo -Wall
```

ANEXO II – Compilação e Ligação

1. Desmembramento e descompactação de arquivo *.tar.gz*

O arquivo *.tar.gz* pode ser desmembrado e descompactado de maneira a gerar, em seu disco, a mesma estrutura de diretórios original dos arquivos que o compõe. Supondo que o arquivo *tar.gz* chame-se "*file.tar.gz*", deve ser utilizado o seguinte comando:

```
tar -zxvf file.tar.gz
```

2. Geração de arquivo *.tar.gz*

Uma estrutura de diretórios existente no disco pode ser completamente copiada e compactada para um arquivo *tar.gz*. Supondo que se deseja copiar o conteúdo do diretório de nome "*dir*", incluindo seus arquivos e subdiretórios, para um único arquivo *tar.gz* de nome "*file.tar.gz*", deve-se, a partir do diretório pai do diretório "*dir*", usar o seguinte comando:

```
tar -zcvf file.tar.gz dir
```