Atividade Prática 02 - Relatório

Guilherme Alves Carvalho

Universidade Federal de Uberlândia - UFU

Resultados

1. **Utilizando dois compiladores diferentes de linguagem C, produza o código assembly para o programa abaixo. Analise as diferenças de cada código assembly produzido em comparação ao código C. A plataforma de SO é de livre escolha.**

```c
#include <stdio.h>
int i = 3;
int j;
main()
{
 int w;
 int z = 3;
 printf("Hello World !\n");
 printf("% d % d % d % d", i, j, w, z);
}
```

- Plataforma escolhida: Ubuntu 20.04.2 LTS

Código assembly gerado pelo gcc

```
 .file "cod.c"
 .text
 .globl  i
 .data
 .align 4
 .type i, @object
 .size i, 4
i:
 .long 3
 .comm j,4,4
 .section  .rodata
.LC0:
 .string "Hello World !"
.LC1:
 .string "% d % d % d % d"
```

```
 .text
 .globl  main
 .type main, @function
main:
.LFB0:
 .cfi_startproc
 endbr64
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset 6, -16
 movq  %rsp, %rbp
 .cfi_def_cfa_register 6
 subq  $16, %rsp
 movl  $3, -8(%rbp)
 leaq  .LC0(%rip), %rdi
 call  puts@PLT
 movl  j(%rip), %edx
 movl  i(%rip), %eax
 movl  -8(%rbp), %esi
 movl  -4(%rbp), %ecx
 movl  %esi, %r8d
 movl  %eax, %esi
 leaq  .LC1(%rip), %rdi
 movl  $0, %eax
 call  printf@PLT
 movl  $0, %eax
 leave
 .cfi_def_cfa 7, 8
 ret
 .cfi_endproc
.LFE0:
 .size main, .-main
 .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
 .section  .note.GNU-stack,"",@progbits
 .section  .note.gnu.property,"a"
 .align 8
```

```
 .long  1f - 0f
 .long  4f - 1f
 .long  5
0:
 .string  "GNU"
1:
 .align 8
 .long  0xc0000002
 .long  3f - 2f
2:
 .long  0x3
3:
 .align 8
4:
```

Código assembly gerado pelo clang

```
        .text
       .file "cod.c"
       .globl  main                    # -- Begin function main
       .p2align  4, 0x90
       .type main,@function
      main:                                   # @main
       .cfi_startproc
      # %bb.0:
       pushq %rbp
       .cfi_def_cfa_offset 16
       .cfi_offset %rbp, -16
       movq  %rsp, %rbp
       .cfi_def_cfa_register %rbp
       subq  $16, %rsp
       movl  $3, -8(%rbp)
       movabsq $.L.str, %rdi
       movb  $0, %al
       callq printf
       movl  i, %esi
```

```
	movl	j, %edx
	movl	-4(%rbp), %ecx
	movl	-8(%rbp), %r8d
	movabsq $.L.str.1, %rdi
	movl	%eax, -12(%rbp)		# 4-byte Spill
	movb	$0, %al
	callq printf
	xorl	%ecx, %ecx
	movl	%eax, -16(%rbp)		# 4-byte Spill
	movl	%ecx, %eax
	addq	$16, %rsp
	popq	%rbp
	.cfi_def_cfa %rsp, 8
	retq
.Lfunc_end0:
	.size main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.type i,@object                 # @i
	.data
	.globl	i
	.p2align	2
i:
	.long 3                         # 0x3
	.size i, 4

	.type .L.str,@object            # @.str
	.section	.rodata.str1.1,"aMS",@progbits,1
.L.str:
	.asciz	"Hello World !\n"
	.size .L.str, 15

	.type .L.str.1,@object          # @.str.1
.L.str.1:
	.asciz	"% d % d % d % d"
	.size .L.str.1, 16
```

```
    .type j,@object                # @j
    .comm j,4,4
    .ident  "clang version 10.0.0-4ubuntu1 "
    .section  ".note.GNU-stack","",@progbits
    .addrsig
    .addrsig_sym printf
    .addrsig_sym i
    .addrsig_sym j
```

- Uma das diferenças perceptíveis em relação aos códigos assembly gerados pelos compiladores gcc e clang a partir do mesmo código fonte são:
    - O compilador do clang inseriu comentários no código assembly, ao contrário do gcc que não inseriu nenhum.
    - A declaração de variáveis globais (.data) pelo compilador clang foram realizadas após a declaração da rotina main, no resultado do gcc elas vieram antes.
    - Ao realizar a chamada da rotina de I/O para mostrar a string “Hello World!”, o clang chamou a rotina printf para tal, já o gcc chamou a rotina puts.

2. **A partir do código-fonte abaixo, gere o programa executável, execute-o e anote a senha que será exibida na tela. Observe que a rotina passcode() não faz parte do programa abaixo e nem da linguagem C. Sua implementação está disponível no arquivo objeto denominado “passcode.o” que acompanha esta lista. Para usar essa rotina é necessário incluir o arquivo de cabeçalho “passcode.h” que acompanha o “passcode.o”. O arquivo “passcode.o” foi criado para funcionar apenas no sistema operacional Linux.**

```c
#include "passcode.h"

main()
{
 char code[11];
 passcode(code);
 printf("% s", code);
}
```

- Foi necessário primeiro gerar o obj do código fonte acima e linkar ambos em um executável
- A senha gerada foi: ABCDEFGHIJ

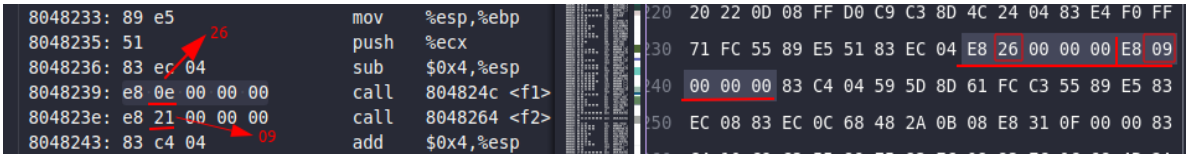
3. **O arquivo de programa "prog01.exe" (no diretório desta lista) possui três funções, main(), f1() e f2(), cujo código fonte em C está listado abaixo. Faça alterações diretamente no arquivo de programa (prog01.exe) para que ao ser executado a função main() chame primeiro f2() e depois f1(). O arquivo de programa prog01.exe foi criado para funcionar apenas no sistema operacional Linux.**

- Resultado

```
valtim@valtim-Aspire-E5-571G:~/Documentos/prog/ufu/
4p/so/introduction/ap2/reverseEng$ ./prog01.exe
F2
F1
```

- Os comandos de análise de arquivos elf (objdump e readelf) foram utilizados para encontrar o local de chamada das respectivas funções no código hexadecimal

- Dessa forma, o endereço de chamada foi alterado, e a f2 foi chamado anteriormente da f1 (as alterações de endereço foram relativas à próxima instrução)



4. **A partir do código-fonte abaixo, crie os arquivos de programas "prog02.exe" e "prog03.exe". Execute cada programa comparando seus tempos de execução. Para isso, utilize o comando "time" (Linux) ou "PowerShell Measure-Command" (Windows). Abaixo exemplos de utilização.**

```c
/* prog02.c */
#include <stdio.h>
main()
{
 int i;
 for (i = 0; i < 10000; i++)
   printf("A\n");
}

/* prog03.c */
#include <stdio.h>
#include <string.h>
main()
{
 int i;
 char str[20001] = "";
 for (i = 0; i < 10000; i += 2)
 {
   *(str + i) = 'A';
   *(str + i + 1) = '\n';
```

```
 }
 *(str + i) = '\0';
 printf("%s", str);
}
```

C:\Users\johndoe>powershell -command "Measure-Command {prog02.exe}"

C:\Users\johndoe>powershell -command "Measure-Command {prog03.exe}"

Obs: Execute 11 vezes cada programa, descarte a primeira execução de cada programa, e tire a média dos demais 10 resultados de cada programa para ter um valor aproximado dos tempos de execução de cada um.

- Foi codificado um shell script bash para automatizar os testes, e o resultado dos testes teve como resultado uma velocidade maior do “prog03.exe”, pelo do mesmo realizar uma única chamada da função “printf” para mostrar a mensagem

```
1   TIME AVERAGE (prog02.exe) => .117851
2   TIME AVERAGE (prog03.exe) => .051757
3   DIFF VALUE: .066094
4
```

- Bash Script

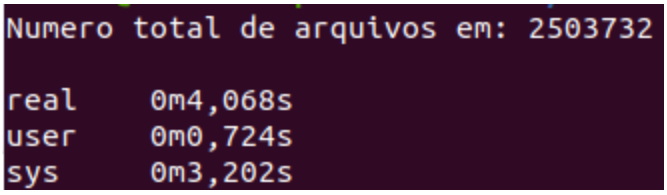
```
#!/usr/bin/env bash
sum=(0 0)
args=("$@")
TIMEFORMAT="%R"
for i in {0..1}
do
 for j in {1..11}
 do
   start=$(date +%s.%6N)
   ./${args[$i]}
```

```
    end=$(date +%s.%6N)
    elapsed=$(echo "scale=6; $end - $start" | bc)
    if [ $j -ne 1 ]
    then
      echo "(${args[$i]}) seconds: $elapsed" >> seconds.txt
      sum[$i]=$(echo "scale=6;${sum[$i]}+$elapsed" | bc)
    fi
  done
  sum[$i]=$(echo "scale=6;${sum[$i]}/10.0" | bc)
  echo "TIME AVERAGE (${args[$i]}) => ${sum[$i]}" >> benchs.txt
done
echo "DIFF VALUE: $(echo "scale=6;${sum[0]}-${sum[1]}" | bc)"
>> benchs.txt
```

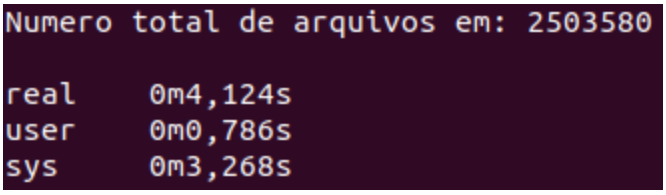
5. **Fazer um programa, em linguagem C, para contar e imprimir o número total de arquivos armazenados em um disco rígido. Implementar e comparar o tempo de execução de três versões desse programa. A primeira versão deve ser programada como um único processo singlethreaded. A segunda versão deve ser programada como múltiplos processos singltehreaded, onde o número de processos (n) deve corresponder ao número de processadores do computador. Caso o computador tenha apenas um processador, então utilize n=2. A terceira versão deve ser programada como múltiplos processos, tal como a segunda versão, contudo cada processo deve utilizar múltiplas threads(mt). O valor de mt deve ser 2. Na segunda e terceira versões, o algoritmo de busca e contagem de arquivos deve ser paralelizado; por exemplo, enquanto um processo conta os arquivos em uma parte do disco (ex. C:\no Windows ou /dev/sda1 no Linux) o outro processo conta os arquivos em outra**

parte (ex. D:\ou /dev/sda2). O mesmo aplica-se para múltiplas threads. A estratégia de paralelização do algoritmo de contagem de arquivos é de livre escolha, assim como a plataforma de SO escolhida para realizar esse exercício.

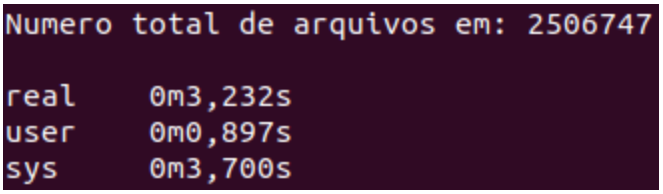
- Usando o comando time, obteve-se os seguintes resultados para cada versão:
    - singlethreaded.c (único processo singlethreaded)



    - multiproc.c (múltiplos processos singlethreaded)



    - procmultithreaded.c (múltiplos processos com duas threads, cada)



- A versão multithreaded foi a mais performática, acredito que elas podem ter melhores resultados se uma estratégia diferente fosse usada para passar os resultados entre os processos, sem utilizar arquivos.

Referências

C compilers. Previous Post:11 Best Free Linux Compilers. Artigo sem referência de DOI.

GCC Assembly output. How do you get assembler output from C/C++ source in gcc?. Artigo sem referência de DOI.

Clang. How to install Clang on Ubuntu. Artigo sem referência de DOI.

gcc-multilib. Problemas de compilação: não é possível encontrar crt1.o. Artigo sem referência de DOI.

Test Automation. Get execution time in seconds – Bytefreaks.net - Bash. Artigo sem referência de DOI.