Lista 5 - Período 2021-1 Remoto - 120 pontos

Nomes: .....

Questão 1 (40 pontos) Seja dado o programa abaixo:

```
#include <stdio.h>
int main(int argc, char *argv[], char *envp[]) {
    int i;
    for (i=0;argv[i] != NULL; i++)
        printf("argv[%2d]: %s\n",i, argv[i]);}
```

a) (5) Crie o arquivo `nome.c`, compile as opções -m32 -fno-PIC -O2 -S para gerar o arquivo `nome.s`. Imprima este arquivo. Comente as linhas do código de montagem. A partir de `nome.s` crie um executável `nome`, rode e imprima a tela de saída do programa.

b) Rode `nome` com `gdb`. Crie um ponto de parada em main. Você terá que usar comandos do gdb para responder às perguntas abaixo. Cada item deverá ser comprovado com a captura da tela dos comandos executados com o gdb. Você pode gerar todos os comandos no gdb necessários para a resposta a todos as perguntas abaixo e apresentar uma única tela de captura. Em cada item, responda o que foi pedido, citando a captura feita.

b1) (5) Documente a sequência de passos usada no gdb, capturando desde o disparo inicial do gdb, run nome e os comandos efetuados no gdb para obter &argc, argc, &argv[0] e argv[0].

b2) (5) Liste em hexa o conteúdo de memória a partir do endereço argv[0], até encontrar byte NULL. Capture a tela do gdb.

b3) (5) Identifique a sequência de caracteres ASCII que corresponde ao conteúdo de memória, usando o comando `print /x` do gdb para criar a tradução em hexa e comprovar que é igual ao valor em memória. Capture a tela do gdb.

c) (10) Altere **um único valor** em `nome.s` (liste e mostre o que foi alterado) e gere `nomem.s` de modo a imprimir todas as variáveis de ambiente. Gere o executável `nomem` a partir de `nomem.s`, rode e imprima a tela, comprovando o sucesso da modificação.

d) (5) Rode `nomem` com `gdb`, indique o valor do endereço apontado por `envp[0]` e liste o endereço de memória a partir deste endereço e até encontrar o byte NULL.

e) (5) Identifique a sequência de caracteres ASCII que corresponde ao conteúdo desta lista de caracteres (string), usando o comando `print /x` do gdb para criar a tradução em hexa e comprovar que é igual ao valor em memória. Capture a tela do gdb.

Questão 2 (30 pontos) Compile o programa abaixo e gere um executável `qfork`.

```
#include <sys/wait.h>
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
int main () {
   int pid ;
   int status;
   pid = fork () ;
```

```
if ( pid < 0 ) { perror ("Erro: "); exit (1) ;}
else if ( pid > 0 ) {
             printf ("\nsou o processo pai, com pid = %d\n", getpid()) ;
             pid = waitpid(pid, &status, 0);
             printf ("pid do processo que liberou o pai = %d\n", pid);
             if (WIFEXITED(status)) printf ("término do filho = %d \n", WEXITSTATUS (status));
             if (WIFSTOPPED(status)) printf ("filho parado= %d \n", WSTOPSIG (status));
             if (WIFSIGNALED(status)) printf ("sinal que causou término do filho = %d \n",
             WTERMSIG (status));
             printf ("Tchau do pai\n");
             exit(0);
             }
         else {  // processo filho
             printf ("sou o processo filho, com pid = %d\n", getpid()) ;
             sleep(10);
             printf("Tchau do filho\n");
             exit(2);}}
```

a) (5) Explique o que o programa faz,considerando todas as possibilidades de quem roda primeiro elevando em conta as interações entre os processos.

b) (5) Você irá rodar este programa em background. Descubra como você pode, a partir da shell, parar o processo filho e depois fazê-lo continuar. Quais sinais são enviados e como são enviados? Imprime uma tela que mostra sua interação com o programa e os comandos executados na shell.

c) (5) Rode o programa em background e pare o processo filho. Agora envie um SIGINT para o filho. O quê acontece? O sinal SIGINT é entregue ao processo filho? Explique.

d) (5) Faça o filho continuar e verifique a saída do programa. Conclua sobre como ocorre a entrega de sinais SIGINT, SIGKILL e SIGSTOP a processos parados.

e) (5) Modifique agora `qfork.c` para que o pai consulte o término do filho, mas de forma não bloqueante. Qual modificação foi feita? Rode algumas vezes o programa e analise o que é impresso pelo pai. Você diria que está correta e coerente a saída impressa pelo programa? Explique. Caso haja inconsistência, que modificação você faria para termos uma saída coerente? Explique sua modificação, se houver.

f) (5) Modifique agora `qfork.c` para que o pai fique em espera pelo término ou parada do filho. Rode o programa em background e veja o que é impresso. Agora, após iniciar a execução, pare o processo filho e documente a saída do programa. Imediatamente ao término do pai, verifique se o programa filho ainda está no sistema. Comente e verifique se há processo zumbi no sistema.

Questão 3 (30 pontos) Para o programa `qfork.c` da questão anterior, foi gerado o código de montagem com opção -m32 -fno-PIC -O2 -S. Algumas linhas não essenciais foram suprimidas, incluindo diretivas para carga do código. Identifique no código as instruções que identificam qual condição garante que as macros WIFEXITED(status), WIFSTOPPED(Status) e WIFTERM(status) são TRUE, além de que bits ou bytes são retornados após as macros WEXITSTATUS(status), WSTOPSIG(status) e WTERMSIG(status), respectivamente. Somente explique as passagens necessárias para justificar sua resposta. O código de montagem será bem longo, mas foque apenas nas linhas que implementam as macros. Apenas estas interessam.

Você deve determinar exatamente as condições dos bytes na variável status que identificam cada uma das três possíveis situações: término normal via exit/return, filho parado por causa de sinal e filho terminado por causa de sinal. Além disso, você tem que apontar os bits ou bytes de status que identificam as causas de término ou os sinais envolvidos. Identifique pelo código de montagem os intervalos válidos para término normal de um processo e os valores possíveis para os sinais que causaram parada ou término do processo. Os valores retornados no código C são apenas exemplos. O que se quer são os valores possíveis que podem ser escolhidos ou identificados pelo SO em virtude do que é de fato analisado pelo código de montagem ao processar as macros correspondentes.

```
.file "qfork.c"
.LC0:
.string "Erro: "
.LC1:
.string "\nsou o processo pai, com pid = %d\n"
.LC2:
.string "pid do processo que liberou o pai = %d\n"
.LC3:
.string "t\303\251rmino do filho = %d \n"
.LC4:
.string "filho parado= %d \n"
.LC5:
.string "sinal que causou t\303\251rmino do filho = %d \n"
.LC6:
.string "Tchau do pai"

.LC7:
.string "sou o processo filho, com pid = %d\n"
.LC8:
.string "Tchau do filho"
main:
1 endbr32
2 leal 4(%esp), %ecx
3 andl $-16, %esp
4 pushl -4(%ecx)
5 pushl %ebp
6 movl %esp, %ebp
7 pushl %ebx
8 pushl %ecx
9 subl $16, %esp
10 movl %gs:20, %eax
11 movl %eax, -12(%ebp)
12 xorl %eax, %eax
13 call fork
14 testl %eax, %eax
15 js .L10
16 je .L3
17 movl %eax, %ebx
18 call getpid
19 pushl %ecx
20 pushl %eax
21 pushl $.LC1
22 pushl $1
23 call __printf_chk
24 addl $12, %esp
25 leal -16(%ebp), %eax
26 pushl $2
27 pushl %eax
28 pushl %ebx
29 call waitpid
30 addl $12, %esp
31 pushl %eax
32 pushl $.LC2
33 pushl $1
34 call __printf_chk
35 movl -16(%ebp), %eax
36 addl $16, %esp
```

```
37 testb $127, %al
38 je .L11
.L4:
39 cmpb $127, %al
40 je .L12
.L5:
41 movl %eax, %edx
42 andl $127, %edx
43 addl $1, %edx
44 subb $1, %dl
45 jle .L6
46 andl $127, %eax
47 pushl %ecx
48 pushl %eax
49 pushl $.LC5
50 pushl $1
51 call __printf_chk
52 addl $16, %esp
.L6:
53 subl $12, %esp
54 pushl $.LC6
55 call puts
56 movl $0, (%esp)
57 call exit
.L3:
58 call getpid
59 pushl %edx
60 pushl %eax
61 pushl $.LC7
62 pushl $1
63 call __printf_chk
64 movl $10, (%esp)
65 call sleep
66 movl $.LC8, (%esp)
67 call puts
68 movl $2, (%esp)
69 call exit
.L10:
70 subl $12, %esp
71 pushl $.LC0
72 call perror
73 movl $1, (%esp)
74 call exit
.L11:
75 movzbl %ah, %eax
76 pushl %edx
77 pushl %eax
78 pushl $.LC3
79 pushl $1
80 call __printf_chk
81 movl -16(%ebp), %eax
82 addl $16, %esp
83 jmp .L4
.L12:
84 movzbl %ah, %eax
85 pushl %ebx
86 pushl %eax
```

```
87 pushl $.LC4
88 pushl $1
89 call __printf_chk
90 movl -16(%ebp), %eax
91 addl $16, %esp
92 jmp .L5
```

Questão 4 (20 pontos) Considere o programa C abaixo.

```
pid_t pid;
sigjmp_buf buf;
int j;
onsidere
void foo (int sig) {
  siglongjmp(buf,getpid());}

void main() {
   sigset_t mask;
   sigemptyset(&mask);
   sigaddset(&mask, SIGUSR1);
   sigprocmask(SIG_UNBLOCK, &mask, NULL);
   signal (SIGUSR1, foo);
   j = sigsetjmp(buf,0);
   printf("%d\n",j);
   pid = fork();
   if (pid==0) {
            printf("%d \n", getpid());
            if (kill (getppid(),SIGUSR1)<0) printf("erro");}
   sleep(2);
   return;}
```

a) (10) Explique a interação entre os processos que serão criados e indique a saída que será impressa, assumindo que o processo inicial tem pid=100 e os processos filhos recebem pid crescente.

b) (10) Considere agora que, no código acima, substituimos `sigsetjmp(buf,0)` por `sigsetjmp(buf,1)`. Qual a implicação desta modificação na interação entre os processos e na saída que será impressa? Se não houver alteração, justifique o porquê.