Procesy 1 - Ustawianie limitów procesu		
Dominik Wróbel	28 III 2019	Czw. 17:00

Spis treści

1 Ustawianie limitów procesu

Zadanie

Proszę pobrać, skompilować i uruchomić program save.c

1.1. Zadania 1,2 i 4

Zadanie 1

Proszę rozbudować program tak aby wartość zmiennej filename mogła być zadana jako (obowiązkowy) argument wywołania programu

Zadanie 2

Proszę rozbudować program tak aby wartość zmiennej s mogła być zadana jako (opcjonalny) argument wywołania programu. Docelowo wywołanie programu save powinno mieć taką składnię: ./save [bytes] file

Zadanie 4

Rozbuduj program o sprawdzanie czy podano wystarczającą liczbę argumentów. Jeżeli nie to zakończ program zwracając 1.

Listing 1: Zadania 1 2 i 4 - plik save.c - poczatek funkcji main

```
...
int main(int argc, char* argv[])
    {

    char* filename;
    int s = 100;
    int fd;

    if(argc == 1 || argc > 3){ // Zadanie 4
          printf("Invalid arguments. Please check options.");
          return 1;
      }
      if(argc == 2){ // Zadanie 1 - one argument -> filename is obligatory
          printf("One argument");
          filename = argv[1];
          fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC);
          if(fd == -1){
              printf("Error opening file");
              exit(EXIT_FAILURE);
          }
      } else if(argc == 3){ // Zadanie 2 - two arguments -> filename is obligatory
          printf("Two arguments");
          filename = argv[2];
          fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC);
          if(fd == -1){
              printf("Error opening file");
              exit(EXIT_FAILURE);
          }
          s = atoi(argv[1]); // second argument is number of bytes
```

```
30            printf("s is %d", s);
31        }
32 ...
```

1.2. Zadanie 3

Zadanie

Proszę przetestować działanie programu:

- Zapisz 100 bajtów do pliku o nazwie tmp1.txt
- Zapisz 53 bajty do pliku o nazwie tmp2.txt

Program działa zgodnie z oczekiwaniami, zapisuje do pliku 100 bajtów (w przypadku braku podania jawnej liczby bajtów) lub przekazaną jako opcje liczbę bajtów.

Listing 2: Testowanie działania save.c

```
1  ~/OperatingSystems/Lab4$ ./save tmp1.txt
2  One argument
3  RLIMIT_FSIZE: cur=-1, max=-1
4  Writing 100 bytes into tmp1.txt file...
5  Returning zero
6
7
8  ~/OperatingSystems/Lab4$ cat tmp1.txt | wc -c
9  100
10
11
12 ~/OperatingSystems/Lab4$ ./save 53 tmp2.txt
13 Two arguments
14 s is 53
15 RLIMIT_FSIZE: cur=-1, max=-1
16 Writing 53 bytes into tmp2.txt file...
17 Returning zero
18
19
20 ~/OperatingSystems/Lab4$ cat tmp2.txt | wc -c
21 53
```

1.3. Zadanie 5

Zadanie 5

Przetestuj działanie:

- Po wykonaniu tej komendy nie powinien pojawić się komunikat FAIL:
  ./save tmp1.txt || echo FAIL
- Po wykonaniu tej komendy powinien pojawić się komunikat OK:
  ./save 200 tmp1.txt && echo OK
- Po wykonaniu tej komendy powinien pojawić się komunikat FAIL:
  ./save || echo FAIL
- Po wykonaniu tej komendy nie powinien pojawić się komunikat OK:
  ./save && echo OK

Listing 3: Testowanie działania save.c w przypadku wywołań z && oraz ||

```
~/OperatingSystems/Lab4$ ./save tmp1.txt || echo FAIL
One argument
RLIMIT_FSIZE: cur=-1, max=-1
Writing 100 bytes into tmp1.txt file...
Returning zero
// nie wyswietla FAIL

~/OperatingSystems/Lab4$ ./save 200 tmp1.txt && echo OK
Two arguments
s is 200RLIMIT_FSIZE: cur=-1, max=-1
Writing 200 bytes into tmp1.txt file...
Returning zero
OK // wyswietla OK


~/OperatingSystems/Lab4$ ./save || echo FAIL
Not enough arguments. Please enter filename.FAIL // wyswietla FAIL


~/OperatingSystems/Lab4$ ./save && echo OK
Not enough arguments. Please enter filename.
// nie wyswietla OK
```

1.4. Zadanie 6

Zadanie 6

Zmodyfikuj funkcje main programu shell ustawiając przed główną pętlą miękki limit maksymalnej wielkości tworzonych plików przez proces (RLIMIT_FSIZE) na wartość 50 bajtów.

Listing 4: plik shell.c - przed główną pętlą

```
...

    struct rlimit r; // ustawianie limitu procesu
    getrlimit(RLIMIT_FSIZE, &r);
    r.rlim_cur = 50;
    setrlimit(RLIMIT_FSIZE, &r);


    while(1){
      printprompt();
      if(readcmd(cmd, MAXCMD) == RESERROR) continue;
      res = parsecmd(cmd, MAXCMD, &cmds);
      printparsedcmds(&cmds);
      executecmds(&cmds);
      dealocate(&cmds);
    }

return 0;

...
```

1.5. Zadanie 7

Zadanie 7

Sprawdź czy ustawiony limit jest dziedziczony przez procesy potomne wykorzystując w tym celu pierwszą aplikację save. Wpisz w swojej konsoli następujące polecenia i zaobserwuj, czy ich działanie jest jak na poniższym listingu.

```
@ ./save 10 tmp3.txt # plik tmp3.txt zostaje utworzony i zawiera 10 bajtów
@ wc -c tmp3.txt
10 tmp.txt
@ ./save tmp4.txt && ./save tmp5.txt # plik tmp4.txt zostaje utworzony i zawiera 50 bajtów,
natomiast plik tmp5.txt nadal nie istnieje
@ wc -c tmp4.txt
50 tmp4.txt
@ wc -c tmp5.txt
wc: tmp5.txt: Nie ma takiego pliku ani katalogu
```

Wynik działania programu zgadza się z listingiem. Ustawiony limit **jest dziedziczony** przez procesy potomne. Wywołania funkcji systemowych takich jak fork czy exec dziedziczą ustawione limity.

Listing 5: plik shell.c - przed główną pętlą

```
@ ./save tmp1.txt || echo FAIL
Parsed command(s):
Command 1:
argv[0]: ./save
argv[1]: tmp1.txt
argv[2]: (null)
Command 1:
argv[0]: echo
argv[1]: FAIL
argv[2]: (null)
One argument
RLIMIT_FSIZE: cur=50, max=-1
Writing 100 bytes into tmp1.txt file...

@ ./save tmp4.txt && ./save tmp5.txt
Parsed command(s):
Command 1:
argv[0]: ./save
argv[1]: tmp4.txt
argv[2]: (null)
Command 1:
argv[0]: ./save
argv[1]: tmp5.txt
argv[2]: (null)
One argument
RLIMIT_FSIZE: cur=50, max=-1
Writing 100 bytes into tmp4.txt file...

@ wc -c tmp4.txt
Parsed command(s):
Command 1:
argv[0]: wc
argv[1]: -c
argv[2]: tmp4.txt
argv[3]: (null)
```

```
50 tmp4.txt

@ wc -c tmp5.txt
Parsed command(s):
Command 1:
argv[0]: wc
argv[1]: -c
argv[2]: tmp5.txt
argv[3]: (null)
wc: tmp5.txt: No such file or directory
```