TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

OPERAČNÉ SYSTÉMY
IPC

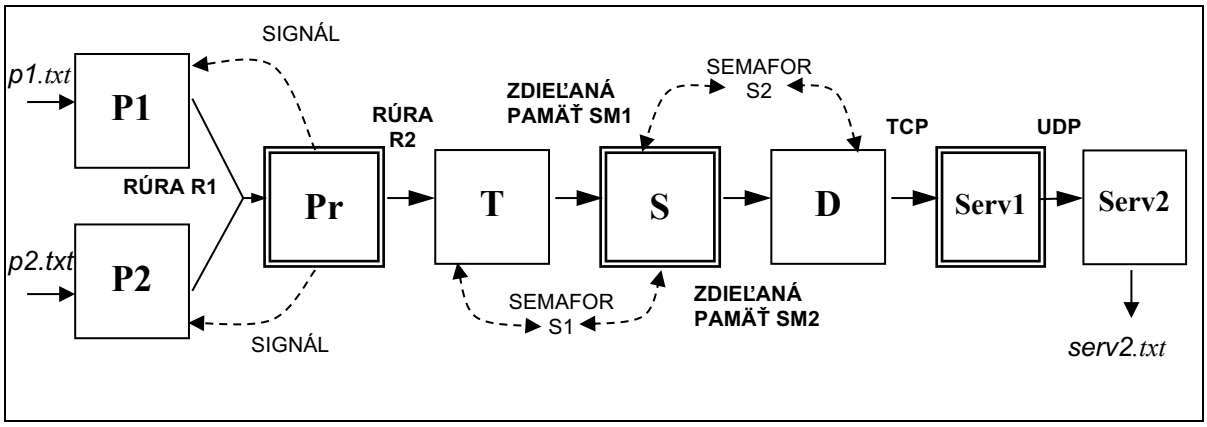
Nikita Zelenkin
Stanislav Voloshyn
Volodymyr Haran
Dmytro Havrysh

Obsah

1. Text zadania

Zadanie Unix1

SCHÉMA



- procesy vyznačené zvýrazneným okrajom sú programy, ktoré budú pri kontrole zadania dodané. Teda treba vypracovať iba programy P1, P2, T, D a Serv2.

POPIS PROCESOV

PROCES Zadanie (vypracovaný študentom, nie je zakreslený v diagrame)

Spúšťanie

```
zadanie <číslo portu 1> <číslo portu 2>
```

Funkcia

Parametre hlavného programu sú čísla portov pre servery Serv1 (TCP) a Serv2 (UDP). Tieto čísla je potrebné týmto procesom odovzdať. Program **Zadanie** nech vyhradí všetky zdroje (pre medziprocesovú komunikáciu) a nech spustí všetky procesy. Všetky procesy nech sú realizované ako samostatné programy.

PROCES Pr

Spúšťanie

```
proc_pr <pid procesu P1> <pid procesu P2>
<id čítaciaho konca rúry R1> <id zapisovacieho konca rúry R2>
```

Funkcia

Proces **Pr** pošle signál SIGUSR1 svojmu hlavnému procesu (Zadanie) na signalizáciu, že je pripravený. Proces **Pr** si údaje bude žiadať:

- Ak pošle signál (SIGUSR1) procesu **P1,** nech tento proces (P1) zapíše slovo prečítané zo súboru *p1.txt*.
- To isté platí aj pre proces **P2** a *p2.txt*. Teda, ak proces **Pr** pošle signál (SIGUSR1) procesu **P2** nech do rúry R1 zapíše slovo proces **P2** zo súboru *p2.txt*.

Proces **Pr** k slovu prijatému z rúry R1 pridá svoju značku a zapíše nové slovo do rúry R2.

PROCES S

Spúšťanie

```
proc_s <id zdieľanej pamäte SM1> <id semaforu S1> <id zdieľanej pamäte SM2>
<id semaforu S2>
```

Funkcia

Proces **S** pošle signál SIGUSR1 svojmu hlavnému procesu (Zadanie) na signalizáciu, že je (proces S) pripravený. Proces **S** príjme slovo zo zdieľanej pamäte SM1 so synchronizáciou semaforom S1 (pozri časť o semaforoch v časti „Popis komunikácie“), pripíše k nemu svoju značku a zapíše ho do zdieľanej pamäte SM2 so synchronizáciou semaforom S2.

PROCES Serv1

Spúšťanie

```
proc_serv1 <číslo portu 1> <číslo portu 2>
```

Funkcia

Proces **Serv1** vytvorí TCP server (na porte <číslo portu 1>), ktorý bude prijímať TCP pakety. Server prijaté slová označí svojou značkou a pošle ich ďalej na UDP server (port <číslo portu 2>). Čísla portov 1 a 2 sú argumenty hlavného procesu (pozri kapitolu „PROCES Zadanie“). Proces **Serv1** pošle signál SIGUSR1 svojmu hlavnému procesu (Zadanie) na signalizáciu, že je (proces Serv1) pripravený. TCP aj UDP server nech vytvorený na lokálnom počítači, teda na počítači „127.0.0.1“ (pozor, nie „localhost“!).

POPIS KOMUNIKÁCIE

Semafor S1

Pre semafor S1 je potrebné vytvoriť dvojicu semaforov. Proces **T** nech sa riadi podľa semaforu S1[0] a proces **S** sa bude riadiť podľa semaforu S1[1], pričom nastavený semafor S1[0] (rozumej nastavený na hodnotu 1) nech znamená, že proces **T** môže zapisovať do zdieľanej pamäte (SM1). Nastavený semafor S1[1] nech znamená, že proces **S** môže zo zdieľanej pamäte (SM1) údaje čítať.

Semafor S2

Pre semafor S2 je potrebné vytvoriť dvojicu semaforov. Proces **S** nech sa riadi podľa semaforu S2[0] a proces **D** sa bude riadiť podľa semaforu S2[1]. Pričom nastavený semafor S2[0] (rozumej nastavený na hodnotu 1) nech znamená, že proces **S** môže zapisovať do zdieľanej pamäte (SM2). Nastavený semafor S2[1] nech znamená, že proces **D** môže zo zdieľanej pamäte (SM2) údaje čítať.

Príklad komunikácie medzi procesom T a procesom S – semafor S1

Pozn.: Pre semafor S2 je to analogické.

Hodnota semaforu	Význam
0	**červená**
1	**zelená**

Proces T sa riadi semaforom S1[0] a proces S sa riadi semaforom S1[1]. To znamená, že kým ma proces T na svojom semafore (S1[0]) **hodnotu 0** (t.j. červená) **– tak stále čaká. To**

isté platí pre proces S. To znamená, že kým proces S má na svojom semafore (S1[1]) **hodnotu 0** (červená) **– to znamená, že musí čakať.**

Semafor treba inicializovať do nasledovného stavu:

S1[0] = 1 (zelená/môžeš) **S1[1] = 0** (červená/stoj)
- tento stav znamená, že proces T môže do zdieľanej pamäte zapisovať. Proces S čaká, má červenú, nemôže čítať.

Proces T má zelenú (teda môže zapisovať)**, lebo má svoj semafor** (S1[0]) **nastavený na hodnotu 1** (zelená)**. Teda do zdieľanej pamäte zapíše svoje údaje. Teraz je potrebné, aby proces T umožnil procesu S údaje čítať. Preto zmení semafory na nasledovný stav:**

[Najprv zmení svoj semafor (semafor S1[0]) z hodnoty **1** na hodnotu **0** (teda zo zelenej na červenú) a semafor procesu S (teda S1[1]) zmení z hodnoty **0** na hodnotu **1** (z červenej na zelenú, aby mu naznačil, že údaje sú zapísané a môže ich teda čítať.)**]**

S1[0] = 0 (červená/stoj) **S1[1] = 1** (zelená/môžeš)
- tento stav znamená, že proces T nemôže do zdieľanej pamäte zapisovať, má čakať. Proces S má zelenú, má v zdieľanej pamäti pripravené údaje a môže údaje z pamäte prečítať.

Teraz už môže semafor S zo zdieľanej pamäte údaje prečítať, lebo jeho semafor (S1[1]) **sa zmenil z hodnoty 0** (červená) **na hodnotu 1** (zelená)**. Prečíta teda údaje zo zdieľanej pamäte a vymení farby semaforov** (t.j. zmení sebe zo zelenej na červenú – z **1** na **0** – a procesu T zmení z červenej na zelenú – z **0** na **1** – aby mohol do pamäte zapisovať)**.**

Súbory p1.txt, p2.txt a serv2.txt
Súbory p1.txt a p2.txt budú obsahovať slová (rozumej reťazce znakov každé v novom riadku), pričom veľkosť slova počas putovania medzi procesmi nepresiahne dĺžku 150 znakov. Súbor serv2.txt nech obsahuje výsledné slová zapísané každé v samostatnom riadku.

Poznámka: Pre účely vývoja programov a testovania je možné programy **Pr**, **S** a **Serv1** stiahnuť zo stránok systému na odovzdávanie zadaní.

Upozornenie:
Pozor, nachádzate sa v paralelnom prostredí! To znamená, že poradie vykonávania procesov nemusí byť rovnaké na rôznych systémoch. Preto dbajte na synchronizáciu a vyhnete sa problémom „predbiehania“ procesov!

„Čo všetko teda potrebujem spraviť??“

1. Potrebujem spraviť programy
   a. **P1** (zdrojový text: **proc_p1.cpp**, spustiteľný program: **proc_p1**),
   b. **P2** (zdrojový text: **proc_p2.cpp**, spustiteľný program: **proc_p2**),
   c. **T** (zdrojový text: **proc_t.cpp**, spustiteľný program: **proc_t**),
   d. **D** (zdrojový text: **proc_d.cpp**, spustiteľný program: **proc_d**),
   e. **Serv2** (zdrojový text: **proc_serv2.cpp**, spustiteľný program: **proc_serv2**) a
   f. **Zadanie** (zdrojový text: **zadanie.cpp**, spustiteľný program: **zadanie**).

   A takisto súbor **makefile** na skompilovanie všetkých zadaní.

2. Musím dodržať názvy programov a ich zdrojových textov uvedené v zátvorkách!
   Názvy súborov zdrojových textov nech sú dodržané tiež.

3. Musím dodržať stanovené názvy vstupných súborov (p1.txt, p2.txt) a výstupného súboru (serv2.txt). Dbať na veľké a malé písmená.

4. Vyhotovené **zdrojové texty** programov spolu so súborom **makefile** (programy Pr, S a Serv1 nie – tie budú pri kontrole dodané) treba zbaliť, najlepšie vo formáte *.tar.gz (napr. zadanie.tar.gz) a odoslať na server.

Príklad súboru makefile

```
all: zadanie proc_p1 proc_p2 proc_t proc_d proc_serv2

zadanie: zadanie.cpp
      g++ zadanie.cpp -o zadanie

proc_p1: ...A...
      ...B...
```

Vyššie je uvedený príklad, ako sa zo zdrojového súboru *zadanie.cpp* vyrobí (skompiluje) hlavný program *zadanie*. Ďalej uveďte, ako sa majú skompilovať ostatné programy. Namiesto časti *...A...* vypíšte ktoré súbory sú potrebné na skompilovanie a vytvorenie programu *proc_p1*. V časti *...B...* uveďte konkrétny kompilačný príkaz, ktorý vyvolá shell, aby vytvoril (skompiloval) program *proc_p1*.

Priebeh kontroly zadania

Pre názornosť a pre predstavu, ako sa vykonáva kontrola je tu uvedený stručný priebeh kontroly odovzdaného zadania:

1. Zavolá sa študentom vytvorený súbor *makefile*, pomocou ktorého sa vytvoria potrebné binárne súbory procesov.
2. Systém nájde a zabezpečí potrebné knižnice na spustenie programov zadania a kontroly.
3. Systém dodá programy *proc_pr*, *proc_s* a *proc_serv1*.
4. Pripravia sa vstupné súbory *p1.txt* a *p2.txt*.
5. Spustí sa hlavný študentom vytvorený program *zadanie*.
6. Po skončení behu celej sústavy procesov sa vyhodnotí súbor *serv2.txt*.

2. Dodefinovanie zadania

K zadaniu sú dodané procesy proc_pr, proc_s, proc_serv1, textové súbory p1.txt a p2.txt. Tieto súbory obsahujú reťazce znakov(slova oddelené znakom "nového riadku").

Súbory:

- proc_p1.cpp
- proc_p2.cpp
- proc_serv2.cpp
- proc_t.cpp
- zadanie.cpp
- proc_d.c
- proc_pr
- proc_s
- proc_serv1
- Makefile
- p1.txt – first input
- p2.txt – second input
- serv2.txt – output

V zadaní treba dorobiť procesy P1, P2, D, T, Serv2 a zadanie.

3. Popis relevantných štruktúr, algoritmov, dátových typov, konštánt

V každom súbore úlohy sme mali buffer, do ktorého sme ukladali údaje a používali ich v nasledujúcich procesoch. Na uchovanie argumentov pre funkcie v nich sme použili aj jednu alebo dve konštanty. Môžete ich vidieť v zdrojovom kóde programu.

Príklad (zadanie.cpp:20):

```
// ***** buffer *****
struct {
    int SEM1, SEM2; // ***** semaphores
    int SHM1, SHM2; // ***** shared memory

    char for_pipe_3[3];
    char PIPE10[20];
    char PIPE20[20];
    char PIPE21[10];

    char P1_char[10];
    char P2_char[10];
    char S1_char[10];
    char S2_char[10];
    char SHM1_char[10];
    char SHM2_char[10];
} buff;
```

```
63
64      ////////////////////////////////////////////////////////////////////////
65      sprintf(buff.for_pipe_3, "%d", pipes.pipefd_1[1]);
66      char *arr_P1[] = {(char*)"proc_p1", buff.for_pipe_3, NULL}, *arr_P2[] = {(char*)"proc_p2", buff.for_pipe_3, NULL};
67
68      // ***** create proc_p1 with argv
69      if ((P1 = fork()) == -1) {
70          perror("fork failed");
71          exit(EXIT_FAILURE);
72      } if (P1 == 0) execve( file: "proc_p1", arr_P1,  envp: NULL);
73      sleep(1);
74
75      // ***** create proc_p2 with argv
76      if ((P2 = fork()) == -1) {
77          perror("fork failed");
78          exit(EXIT_FAILURE);
79      } if (P2 == 0) execve( file: "proc_p2", arr_P2,  envp: NULL);
80      sleep(1);
81
82      ////////////////////////////////////////////////////////////////////////
83
```

(proc_serv2.cpp:22):

```
int socketfd;
int txt;
int len_client;
int check;
char buff[151];
```

4. Analýza problematiky

Každý problém sme riešili osobitne:

a) Po ukončení procesu d sme použili službu **shmctl()** a **semctl()** s potrebnými parametrami
b) Vyskytol sa problém, ktorý sme okamžite pripojili k serveru, a potom sa práve vytvoril. A problém s portami, ktorý sme vyriešili použitím rôznych portov
c) Keď sme sa dozvedeli, že **proc_s** je nekonečný, odstránili sme ho príkazom **kill()**

5. Popis navrhovaného riešenia:

Spôsob synchronizácie nastal prostredníctvom príkazu **sleep()**.

6. Záver a zhodnotenie

Počas celého zadania sme mali problémy:

a) čisteníe a prideľovaníe pamäte a odstraňovanie semaforov
b) vytváraním správnych kľúčov
c) pripojenie k serveru
d) bol problem z **proc_s**

Stratilo sa veľa času :)
Problémov bolo ešte viac

7. Použitá literatúra a informačné zdroje

[1]. Знакомство с межпроцессным взаимодействием на Linux, [Dátum: 18.6.2011], [Dátum citovanie: 3.11.2015], Moskva – Rusko, EDISON s.r.o.
https://habr.com/ru/post/122108/

[2]. Roman Brovko, Программирование Linux (IPC), [Dátum: 2016]
https://www.youtube.com/watch?v=z4UUO64oKLs&list=PLwwk4BHih4fgXqxB-T-0kb8gGHXiP73n1