Programowanie współbieżne

Laboratorium 7: Pamięć współdzielona i semafory

Kamil Zdeb, nr albumu: 235871

Piątek TN 9:15-12:15

05.06.2020

1 Wstęp

Programy miały być uruchamiane na komputerze z systemem Linux.

2 Realizacja zadań

Programy wykonałem na posiadanej na swoim komputerze dystrybucji Linux Manjaro przy użyciu edytora *Sublime Text*. W celu sprawniejszej pracy użyłem narzędzia *make* ułatwiającego kompilację plików.

2.1 Zadanie 1: Problem producenta i konsumenta

Zadanie to składa się z trzech programów. Pierwszy, czyli init tworzy pamięć współdzieloną i semafory, kolejny obsługuje konsumenta, a trzeci producenta. Program init wywołuje się bezparametrowo. Program konsumenta wywołuje się z dwoma parametrami(identyfikator producenta, liczba odbieranych towarów), a program producenta z dwoma parametrami(identyfikator producenta, liczba produkowanych towarów).

2.1.1 Kod programu

Program init tworzy semafory i pamięć współdzieloną.

```
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define BSIZE 4 // Rozmiar bufora
#define LSIZE 80 // Dlugosc linii

typedef struct {
  char buf[BSIZE][LSIZE];
  int head;
  int tail;
  int cnt;
  sem_t mutex;
  sem_t empty;
  sem_t full;
} bufor_t;


int main(int argc,char *argv[]) {
  int i,stat,k, pid, size, fd,res;
  bufor_t *wbuf ;
  char c;
// Utworzenie segmentu ----------------------------
  shm_unlink("bufor");
  fd=shm_open("bufor", O_RDWR|O_CREAT , 0774);
  if(fd == -1){
```

```
    perror("open"); _exit(-1);
  }
  printf("fd: %d\n",fd);
  size = ftruncate(fd,sizeof(bufor_t));
  if(size < 0) {
    perror("trunc");
    _exit(-1);
  }
// Odwzorowanie segmentu fd w obszar pamieci procesow
    wbuf = (bufor_t *)mmap(0, sizeof(bufor_t), PROT_READ|PROT_WRITE
    ,MAP_SHARED, fd, 0);
  if(wbuf == NULL) {
    perror("map");
    _exit(-1);
  }
// Inicjacja obszaru --------------------------------
  wbuf-> cnt = 0;
  wbuf->head = 0;
  wbuf->tail = 0;
  if(sem_init(&(wbuf->mutex),1,1)){
    perror("mutex");
    _exit(0);
  }
  if(sem_init(&(wbuf->empty),1,BSIZE)) {
    perror("empty"); _exit(0);
  }
  if(sem_init(&(wbuf->full),1,0)) {
    perror("full");
    _exit(0);
  }
  sem_close(&(wbuf->mutex));
  sem_close(&(wbuf->empty));
  sem_close(&(wbuf->full));
  return 0;
}
```

Program konsumenta odbiera dane z pamięci współdzielonej

```
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define BSIZE 4 // Rozmiar bufora
#define LSIZE 80 // Dlugosc linii

typedef struct {
 char buf[BSIZE][LSIZE];
 int head;
 int tail;
 int cnt;
 sem_t mutex;
 sem_t empty;
 sem_t full;
} bufor_t;

```

```
21
22 int main(int argc,char *argv[]) {
23   int i,stat,k, pid, size, fd,res;
24   bufor_t *wbuf ;
25   char c;
26   int number = atoi(argv[1]);
27   int steps = atoi(argv[2]);
28 // Utworzenie segmentu ---------------------------
29   fd=shm_open("bufor", O_RDWR , 0774);
30   if(fd == -1){
31     perror("open");
32     _exit(-1);
33   }
34   printf("fd: %d\n",fd);
35   wbuf = (bufor_t *)mmap(0, sizeof(bufor_t) ,PROT_READ|PROT_WRITE,
       MAP_SHARED, fd, 0);
36   if(wbuf == NULL) {
37     perror("map");
38     _exit(-1);
39   }
40   // Konsument
41   for(int i=0;i<steps;i++) {
42     sem_wait(&(wbuf->full));
43     sem_wait(&(wbuf->mutex));
44     printf("Konsument %d - cnt: %d odebrano %s\n", number,
45     wbuf->cnt, wbuf->buf[wbuf->tail]);
46     wbuf-> cnt --;
47     wbuf->tail = (wbuf->tail +1) % BSIZE;
48     sem_post(&(wbuf->mutex));
49     sem_post(&(wbuf->empty));
50     sleep(1);
51   }
52   _exit(i);
53
54 }
```

Program producenta wysyła dane do pamięci współdzielonej.

```
1 #include <sys/mman.h>
2 #include <fcntl.h>
3 #include <semaphore.h>
4 #include <stdio.h>
5 #include <unistd.h>
6 #include <sys/types.h>
7 #include <sys/wait.h>
8 #define BSIZE 4 // Rozmiar bufora
9 #define LSIZE 80 // Dlugosc linii
10
11 typedef struct {
12  char buf[BSIZE][LSIZE];
13  int head;
14  int tail;
15  int cnt;
16  sem_t mutex;
17  sem_t empty;
18  sem_t full;
19 } bufor_t;
20
```

```c
int main(int argc,char *argv[]) {
  int i,stat,k, pid, size, fd,res;
  int number = atoi(argv[1]);
  int steps = atoi(argv[2]);
  bufor_t *wbuf ;
  char c;
// Utworzenie segmentu ---------------------------
  fd=shm_open("bufor", O_RDWR , 0774);
  if(fd == -1){
    perror("open");
    _exit(-1);
  }
  printf("fd: %d\n",fd);
  wbuf = (bufor_t *)mmap(0, sizeof(bufor_t) ,PROT_READ|PROT_WRITE,
    MAP_SHARED, fd, 0);
  if(wbuf == NULL) {
    perror("map");
    _exit(-1);
  }
  // Producent
  for(i=0;i<steps; i++) {
    // printf("Producent: %i\n",i);
    sem_wait(&(wbuf->empty));
    sem_wait(&(wbuf->mutex));
    sprintf(wbuf->buf[wbuf->head],"Komunikat %d",i);
    printf("Producent %d - cnt:%d head: %d tail: %d\n",
    number,
    wbuf-> cnt,wbuf->head,wbuf->tail);
    wbuf-> cnt ++;
    wbuf->head = (wbuf->head +1) % BSIZE;
    sem_post(&(wbuf->mutex));
    sem_post(&(wbuf->full));
    sleep(1);
  }
  _exit(i);
}
```

2.2 Zadanie 2a: Znajdowanie liczb pierwszych - wersja bez semaforów

Program stworzyłem w oparciu o jeden z programów z którychś poprzednich zajęć. Program korzysta jedynie z pamięci współdzielonej.

Rysunek 1: Efekt wykonania się programu

```
[kamil@kamil-pc lab7]$ ./zad2a 2 1000 10
fd: 3
Przedział'2-101, znaleziono 29 liczb pierwszych
Przedział'101-200, znaleziono 23 liczb pierwszych
Przedział'200-299, znaleziono 17 liczb pierwszych
Przedział'299-398, znaleziono 17 liczb pierwszych
Przedział'398-497, znaleziono 16 liczb pierwszych
Przedział'497-596, znaleziono 15 liczb pierwszych
Przedział'596-695, znaleziono 17 liczb pierwszych
Przedział'695-794, znaleziono 13 liczb pierwszych
Przedział'794-893, znaleziono 17 liczb pierwszych
Przedział'893-992, znaleziono 14 liczb pierwszych
Time elapsed: 0.001916 seconds
Number of primes: 178
```

2.2.1 Kod programu

Poniższy program tworzy pamięć współdzieloną, następnie dzieli przedział na podprzedziały. Następnie dla każdego podprzedziału obliczana jest liczba liczb pierwszych, która jest zapisywana w strukturze pamięci współdzielonej. Następnie proces macierzysty czeka na zakończenie procesów potomnych, a gdy to nastąpi odczytuje dane z pamięci współdzielonej oraz sumuje liczby znalezionych liczb pierwszych.

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <math.h>
6  #include <tgmath.h>
7  #include <time.h>
8  #include <errno.h>
9  #include <fcntl.h>
10 #include <mqueue.h>
11 #include <sys/types.h>
12 #include <sys/wait.h>
13 #include <sys/mman.h>
14
15 int prime(int n);
16
17 void main(int argc, char * argv[]){
18   int bounds = atoi(argv[3]);
19   typedef struct {
20     int pocz;
21     int kon ;
22     int suma;
23   } dane_t;
24
25   typedef struct {
26     int wymiar;
27     dane_t dane[bounds];
```

```
28   }buf_t;
29
30   buf_t * buf;
31   struct timespec start, finish;
32   double time_elapsed;
33   clock_gettime(CLOCK_MONOTONIC, &start);
34   int status;
35   int lower_bound = atoi(argv[1]);
36   int upper_bound = atoi(argv[2]);
37   int size;
38   int pids[bounds];
39     int number_of_primes = 0;
40   int bound = (upper_bound-lower_bound)/bounds;
41   int writer;
42   int fd;
43   int res;
44
45   shm_unlink("bufor");
46   fd=shm_open("bufor", O_RDWR|O_CREAT , 0774);
47   if(fd == -1){
48     perror("open"); _exit(-1);
49   }
50   printf("fd: %d\n",fd);
51   size = ftruncate(fd,sizeof(buf_t));
52   if(size < 0) {
53     perror("trunc");
54     _exit(-1);
55   }
56   int lb[bounds];
57   int ub[bounds];
58 // Odwzorowanie segmentu fd w obszar pamieci procesow
59   buf = (buf_t *)mmap(0, sizeof(buf_t), PROT_READ|PROT_WRITE,
       MAP_SHARED, fd, 0);
60
61     for(int i=0 ; i<bounds; i++){
62       lb[i] = lower_bound+i*bound;
63             ub[i] = lower_bound+(i+1)*bound;
64       //printf("Wys ano przedzia  (%d-%d)\n", lb, ub);
65     }
66     for(int i=0; i<bounds; i++){
67       if((pids[i] = fork()) == 0) { /* Proces potomny ---*/
68         dane_t data;
69             int primes_found = 0;
70         //printf("Odebrano przedzia  (%d-%d)\n", lower_bound,
       upper_bound);
71
72           for(int j=lb[i]; j<ub[i]; j++){
73               primes_found += prime(j);
74           }
75           data.pocz = lb[i];
76           data.kon = ub[i];
77           data.suma = primes_found;
78           buf->dane[i] = data;
79           return(0);
80       }
81     }
82  /* Proces macierzysty */
```

```
83
84    for(int i=0; i<bounds; i++){
85      pids[i] += wait(&status);
86          WEXITSTATUS(status);
87    }
88    for(int i=0 ; i<bounds; i++){
89    printf("Przedzia '%d-%d, znaleziono %d liczb pierwszych\n", buf
        ->dane[i].pocz, buf->dane[i].kon, buf->dane[i].suma);
90    number_of_primes+=buf->dane[i].suma;
91    //printf("%d\n", bytes_read);
92    }
93    //writer += wait(&status);
94      //WEXITSTATUS(status);
95    clock_gettime(CLOCK_MONOTONIC, &finish);
96    time_elapsed = (finish.tv_sec - start.tv_sec);
97    time_elapsed += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;
98    printf("Time elapsed: %f seconds\n", time_elapsed);
99      printf("Number of primes: %d \n", number_of_primes);
100     return(0);
101 }
102
103 int prime(int n){
104     for(int j=2; j*j<n; j++){
105             if(n%j==0) return(0);
106         }
107     return(1);
108 }
```

2.3 Zadanie 2b: Znajdowanie liczb pierwszych - wersja z semaforami

Rysunek 2: Efekt wykonania się programu

```
[kamil@kamil-pc lab7]$ ./zad2b 2 1000 10
fd: 3
Przedział'2-101, znaleziono 29 liczb pierwszych
Przedział'101-200, znaleziono 23 liczb pierwszych
Przedział'200-299, znaleziono 17 liczb pierwszych
Przedział'299-398, znaleziono 17 liczb pierwszych
Przedział'398-497, znaleziono 16 liczb pierwszych
Przedział'596-695, znaleziono 17 liczb pierwszych
Przedział'497-596, znaleziono 15 liczb pierwszych
Przedział'695-794, znaleziono 13 liczb pierwszych
Przedział'794-893, znaleziono 17 liczb pierwszych
Przedział'893-992, znaleziono 14 liczb pierwszych
Time elapsed: 0.002444 seconds
Number of primes: 178
```

2.3.1 Kod programu

Poniższy program tworzy pamięć współdzieloną oraz semafory z odpowiednimi zmiennymi, następnie dzieli przedział na podprzedziały. Różni się od poprzedniego tym, że w buforze jest miejsce na mniej przedziałów niż istnieje w programie. Następnie dla każdego podprzedziału obliczana jest liczba liczb pierwszych, która jest zapisywana w strukturze pamięci współdzielonej. Proces potomny rozpocznie właściwą część swojego działania dopiero wtedy kiedy w buforze będzie miejsce na dane. Proces macierzysty sprawdza czy w buforze znajdują się jakieś dane. Jeśli nie to na nie czeka. W momencie odczytania danych sumuje liczbę znalezionych liczb pierwszych.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <tgmath.h>
#include <time.h>
#include <errno.h>
#include <fcntl.h>
#include <mqueue.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <semaphore.h>
#define SIZE 4

int prime(int n);

void main(int argc, char * argv[]){
  int bounds = atoi(argv[3]);
  typedef struct {
    int pocz;
    int kon ;
    int suma;
  } dane_t;

  typedef struct {
    int wymiar;
    dane_t dane[SIZE];
    int head;
    int tail;
    int cnt;
    sem_t mutex;
    sem_t empty;
    sem_t full;
  }buf_t;

  buf_t * buf;
  struct timespec start, finish;
  double time_elapsed;
  clock_gettime(CLOCK_MONOTONIC, &start);
  int status;
  int lower_bound = atoi(argv[1]);
  int upper_bound = atoi(argv[2]);
  int size;
```

```
45   int pids[bounds];
46     int number_of_primes = 0;
47   int bound = (upper_bound-lower_bound)/bounds;
48   int writer;
49   int fd;
50   int res;
51
52   shm_unlink("bufor");
53   fd=shm_open("bufor", O_RDWR|O_CREAT , 0774);
54   if(fd == -1){
55     perror("open"); _exit(-1);
56   }
57   printf("fd: %d\n",fd);
58   size = ftruncate(fd,sizeof(buf_t));
59   if(size < 0) {
60     perror("trunc");
61     _exit(-1);
62   }
63   int lb[bounds];
64   int ub[bounds];
65 // Odwzorowanie segmentu fd w obszar pamieci procesow
66   buf = (buf_t *)mmap(0, sizeof(buf_t), PROT_READ|PROT_WRITE,
       MAP_SHARED, fd, 0);
67
68   buf-> cnt = 0;
69   buf->head = 0;
70   buf->tail = 0;
71   if(sem_init(&(buf->mutex),1,1)){
72     perror("mutex");
73     _exit(0);
74   }
75   if(sem_init(&(buf->empty),1,SIZE)) {
76     perror("empty"); _exit(0);
77   }
78   if(sem_init(&(buf->full),1,0)) {
79     perror("full");
80     _exit(0);
81   }
82
83     for(int i=0 ; i<bounds; i++){
84       lb[i] = lower_bound+i*bound;
85             ub[i] = lower_bound+(i+1)*bound;
86       //printf("Wys ano przedzia  (%d-%d)\n", lb, ub);
87     }
88     for(int i=0; i<bounds; i++){
89       if((pids[i] = fork()) == 0) { /* Proces potomny ---*/
90         sem_wait(&(buf->empty));
91         sem_wait(&(buf->mutex));
92         dane_t data;
93             int primes_found = 0;
94
95           for(int j=lb[i]; j<ub[i]; j++){
96               primes_found += prime(j);
97           }
98           data.pocz = lb[i];
99           data.kon = ub[i];
100          data.suma = primes_found;
```

```
101             //printf("Wys ano %d-%d\n", lb[i], ub[i]);
102             buf->dane[buf->head] = data;
103       buf-> cnt ++;
104       buf->head = (buf->head +1) % SIZE;
105       sem_post(&(buf->mutex));
106       sem_post(&(buf->full));
107             return(0);
108       }
109     }
110  /* Proces macierzysty */
111   for(int i=0 ; i<bounds; i++){
112     sem_wait(&(buf->full));
113     sem_wait(&(buf->mutex));
114
115     printf("Przedzia '%d-%d, znaleziono %d liczb pierwszych\n",
      buf->dane[buf->tail].pocz, buf->dane[buf->tail].kon, buf->dane[
      buf->tail].suma);
116     number_of_primes+=buf->dane[buf->tail].suma;
117     buf-> cnt --;
118     buf->tail = (buf->tail +1) % SIZE;
119     sem_post(&(buf->mutex));
120     sem_post(&(buf->empty));
121
122   //printf("%d\n", bytes_read);
123   }
124   for(int i=0; i<bounds; i++){
125     pids[i] += wait(&status);
126         WEXITSTATUS(status);
127   }
128   //writer += wait(&status);
129     //WEXITSTATUS(status);
130   clock_gettime(CLOCK_MONOTONIC, &finish);
131   time_elapsed = (finish.tv_sec - start.tv_sec);
132   time_elapsed += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;
133   printf("Time elapsed: %f seconds\n", time_elapsed);
134     printf("Number of primes: %d \n", number_of_primes);
135     return(0);
136 }
137
138 int prime(int n){
139     for(int j=2; j*j<n; j++){
140             if(n%j==0) return(0);
141         }
142     return(1);
143 }
```