Operativni sistemi 2

1. Za operativni sistem Linux, napisati program na programskom jeziku C koji treba da iz N paralelnih procesa ispiše redom brojeve od 0 do M*N-1, tako da apsolutna razlika uzastopno ispisanih brojeva iz jednog procesa bude N. Dakle, jedan proces treba da ispiše brojeve 0, N, 2N,...; drugi proces ispisuje brojeve 1, N+1, 2N+1, itd. Za potrebe komunikacije i sinhronizacije između procesa dozvoljeno je koristiti isključivo semafore. Svi alocirani resursi moraju biti oslobođeni.

Rešenje:

```
#include<stdio.h>
#include<sys/sem.h>
#include<stdlib.h>
#include <unistd.h>

#define N 5
#define M 5

int main(){
	int i, j, sem_id;
	sem_id = semget((key_t)123, N, 0666 | IPC_CREAT);
	struct sembuf sem;
	for(i=0;i<N;i++){
		if(fork()==0){
			for(j=0;j<M;j++){
				//sem[i].wait();
				sem.sem_num = i;
				sem.sem_op = -1;
				sem.sem_flg = SEM_UNDO;
				semop(sem_id, &sem, 1);

				printf("%d\n",i+j*N);

				//sem[(i+1) % N].signal()
				sem.sem_num = (i+1)%N;
				sem.sem_op = 1;
				sem.sem_flg = SEM_UNDO;
				semop(sem_id, &sem, 1);
			}
			if (i==N-1) {
				semctl(sem_id, 0, IPC_RMID);
			}
			exit(0);
		}
	}
	//sem[0].signal();
	sem.sem_num = 0;
	sem.sem_op = 1;
	sem.sem_flg = SEM_UNDO;
	semop(sem_id, &sem, 1);
}
```

2. Neki programer je rešavao sledeći problem. Potrebno je izračunati prvih N elemenata nekog reda koji je zadat sa: $a_0 = 5$ i $a_i = f(a_{i-1})$. Funkcija f u svom radu koristi samo prosleđeni parametar i lokalne promenljive. Rešenje koje je ponudio programer je sledeće:

```
void main(){
      int zeton = 0;
      int cevi[N+1][2];
      int i;
      int a[N];
      a[0] = 5;
      for (i=0;i<N;i++) pipe(cevi[i]);
      for (i=1;i<N;i++){
            if (fork() == 0){
                  read(cevi[i][IN], &zeton, sizeof(int));
                  a[i] = f(a[i-1]);
                  printf("%d\n",a[i]);
                  write(cevi[i+1][OUT], &zeton, sizeof(int));
                  exit(0);
            }
      }
      write(cevi[1][OUT], &zeton, sizeof(int));
}
```

Dato rešenje ima više nedostataka koji ne utiču na ispravnost ispisa i kao takve ih ne treba uzimati u obzir pri obrazlaganju odgovora na sledeće pitanje. Da li će program ispisati korektne vrednosti (a_1, a_2,...,a_{N-1})? Odgovor kratko i precizno obrazložiti.

Odgovor:
Ne. Programer je ispravno sinhronizovao računanja, tako da se element a_i uvek računa tek pošto je izračunat element a_{i-1}, ali je previdео da se te vrednosti izračunavaju u različitim adresnim prostorima, te da će pri izračunavanju elementa a_i biti korišćena neka slučajna vrednost iz niza a umesto prethodno izračunate vrednosti a_{i-1}.

Napomena: Šta je potrebno izmeniti kako bi program ispisivao korektne vrednosti?

3. Implementirati funkciju `void mergeSort(int n, int m, int niz[])` koja sortira niz od *n* elemenata u rastućem redosledu na sledeći način. Funkcija uzima po *m* elemenata niza i sortira ih u konkurentnim procesima pomoću algoritma datog funkcijom `int sort(int n, int a[])`, a zatim tako sortirane podnizove spaja (*engl.* merge sort). Na raspolaganju su sledeće funkcije za rad sa imenovanim cevovodima (*engl.* Named Pipes):

```
// Upisuje elemente niza a, duzine n u cevovod pipeName
void sendToPipe(char* pipeName, int n, int a[]);
// Čita niz elemenata a iz cevovoda pipeName i njihov broj broj smešta u *n
void readFromPipe(char* pipeName, int* n, int a[]);
// Vraća ime sledećeg cevovoda koji može da se koristi
char* getNextPipeName();
// Obezbeđuje da funkcija getNextPipeName vraća imena cevovoda od početka
void resetPipeNames();
```

Rešenje:

```
int mergeSort(int n, int m, int niz[]){
  // Creating child processes
  int numOfChildren = 0;
  for (int i = 0; i < n; i += m){
      numOfChildren++;
      char* childPipeName = getNextPipeName();
      int len = (i + m < n) ? m : n - i;
      int pid = fork();
      if (pid == 0){          //child
        sort(len, niz + i);
        sendToPipe(childPipeName, len, niz + i);
        exit(0);
      }
  }

  // Collecting sorted subarrays
  int** b = new int [numOfChildren];
  int* cntrs = new int [numOfChildren];
  int* sizes = new int [numOfChildren];

  resetPipeNames();
  for(int c = 0; c < numOfChildren; ++c) {
    b[c] = new int[m];
    cntrs[c] = 0;
    readFromPipe(getNextPipeName(), &sizes[c], b[c]);
  }

  // Merging
  int cntr = 0;
  while (cntr < n) {
    int arrayInd = -1, min = -1;
    for (int i = 0; i < numOfChildren; ++i)
      if (cntrs[i] < sizes[i] && (arrayInd == -1 || min > b[i][cntrs[i]])){
        min = b[i][cntrs[i]];
        arrayInd = i;
      }
      niz[cntr++] = min;
      cntrs[arrayInd]++;
  }



  // Release temporary space
  for (int i = 0; i < numOfChildren; i++)
    delete b[i];
  delete b;
  delete cntrs;
  delete sizes;
  return 0;
}
```

4. Kolokvijum 3 - Januar 2012

Na jeziku C/C++, koristeći mehanizam prosleđivanja poruka operativnog sistema Linux, dati rešenje problema filozofa koji večeraju (*dining philosophers*), pri čemu je data funkcija `philosopher` koja kao argument prima jedinstveni broj (identifikator) filozofa. Svaki filozof pri slanju zahteva identifikuje se ovim brojem. Takođe, navedena je struktura poruka koje se razmenjuju, kao i značenje vrednosti svakog polja.

```
#define MESSAGE_Q_KEY 1

struct requestMsg {
 long mtype;  // tip poruke – identifikator filozofa
 char msg[1]; // operacija - vrednost: 1 - request forks, 2 - release forks
};

void philosopher(int id) {
 int requestMsgQueueId = msgget(MESSAGE_Q_KEY, IPC_CREAT | 0666);
 int responseMsgQueueId = msgget(MESSAGE_Q_KEY + 1, IPC_CREAT | 0666);
 size_t len = sizeof(char);

 while (1) {
  //request forks
  struct requestMsg msg;
  msg.mtype = id;
  msg.msg[0] = (char) 1;
  msgsnd(requestMsgQueueId, &msg, len, 0);
  msgrcv(responseMsgQueueId, &msg, len, id, 0);

  //eat
  sleep(1);

  //release forks
  msg.mtype = id;
  msg.msg[0] = (char) 2;
  msgsnd(requestMsgQueueId, &msg, len, 0);

  //think
  sleep(1);
 }
}
```

Napisati implementaciju centralnog procesa koji na početku nad funkcijom `philosopher` kreira potreban broj filozofa predstavljenih procesima, a zatim u vidu konobara (*waiter*) komunicira sa filozofima i obezbeđuje njihovu sinhronizaciju. Nije potrebno proveravati uspešnost izvršavanja operacije nad sandučićima (*message queue*).

Rešenje:

```
void acquireForksForPhilosopher(int *forks[N], int id, int msgQueueId) {
 forks[id] = 0;
 forks[(id + 1) % N] = 0;
 struct requestMsg msg_buf;
 msg_buf.mtype = id + 1;
 msg_buf.msg[0] = 1;
 msgsnd(msgQueueId, &msg_buf, sizeof(char), 0);
}

int main() {
 int requestMsgQueueId = msgget(MESSAGE_Q_KEY, IPC_CREAT | 0666);
 int responseMsgQueueId = msgget(MESSAGE_Q_KEY + 1, IPC_CREAT | 0666);
 size_t len = sizeof(char);

 //philosophers
 int id;
 for (id = 1; id <= N; id++) { // rezervisana vrednost za mtype=0
  if (fork() == 0) {
   philosopher(id);
  }
```

```
 }

 //waiter
 int *forks[N], *requests[N];
 for (id = 0; id < N; id++) {
  forks[id] = 1;
  requests[id] = 0;
 }

 struct requestMsg msg_buf;
 while (1) {
  int r = msgrcv(requestMsgQueueId, &msg_buf, len, 0, 0);
  id = (int) msg_buf.mtype - 1;

  if (msg_buf.msg[0] == 1) { //request forks
   if (forks[id] && forks[(id + 1) % N]) {
    acquireForksForPhilosopher(forks, id, responseMsgQueueId);
   } else {
    requests[id] = 1;
   }
  } else { //Release forks
   forks[id] = 1;
   forks[(id + 1) % N] = 1;

   // check neighbors
   int leftNeighbour = id ? id - 1 : N - 1;
   int rightNeighbour = (id + 1) % N;
   if (requests[rightNeighbour] && forks[(rightNeighbour + 1) % N]) {
    requests[rightNeighbour] = 0;
    acquireForksForPhilosopher(forks, rightNeighbour,
      responseMsgQueueId);
   }
   if (requests[leftNeighbour] && forks[leftNeighbour]) {
    requests[leftNeighbour] = 0;
    acquireForksForPhilosopher(forks, leftNeighbour,
      responseMsgQueueId);
   }
  }
 }
}
```

Eclipse debugging: Kako bi bilo moguće pratiti proces potomak i proces dete prilikom njihovog izvršavanja, u tekućem direktorijumu projekta u fajlu *.gdbinit* dodati sledeću liniju

```
set detach-on-fork off
```

i/ili

```
follow-fork-mode child
```

Više detalja:

http://dirkraffel.com/2008/06/27/debugging-multiple-processes-with-eclipse-cdt/

http://sourceware.org/gdb/onlinedocs/gdb/Forks.html#Forks

5. Tekst fajl `config` u svakom svom redu sadrži ime tačno jednog programa. Napisati program na jeziku C/C++ za Linux, koji će čitati taj fajl i za program iz svakog reda kreirati tačno jedan proces. Zatim startni program treba da se odmah završi, ne čekajući da njegovi potomci završe. Bilo kakva greška treba samo da prekine ovaj program. Pretpostaviti da se svi potrebni fajlovi nalaze u istom direktorijumu i da imena programa u fajlovima neće imati više od 1000 znakova.

Rešenje:

```
#include <stdio.h>
#include <unistd.h>

int main(){
  FILE *in = fopen("config", "r");

  if (in == NULL) { return -1; }

  char program[1001];
  while (fscanf(in, "%s", program) > 0){  //cita niz karaktera do prvog
                                          //blanko znaka
      int PID = fork();

      if (PID == -1) { return -1; }
      else if (PID == 0) {                //child
            execl(program, NULL);
            exit(0);
      }
  }

  fclose(in);
  return 0;
}
```