Drugi kolokvijum iz Operativnih sistema 1
Maj 2022.

1. (10 poena)

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

const int M = ..., N = ...;
extern int mat[M][N];

int max (int i) {
  int m = mat[i][0];
  for (int j=1; j<N; j++)
    if (mat[i][j]>m) m = mat[i][j];
  return m;
}


int main () {

  for (int i=0; i<M; i++) {
    pid_t pid = fork();
    if (pid<0) {
      printf("Error: Cannot create a child process.\n");
      exit(-1);
    }
    if (pid==0)
      exit(max(i));
  }

  int max;
  wait(&max);
  for (int i=1; i<M; i++) {
    int m;
    wait(&m);
    if (m>max) max = m;
  }
  printf("Max: %d\n",max);
  exit(0);
}
```

2. (10 poena)

```
void yield () {
  asm {
  ; Save the current context
  push      spc ; save regs on the process stack
  push      ssp
  push      spsw
  push      r0
  push      r1
  ...
  push      r31
  load      r0, oldRrunning ; r0 now points to the running PCB
  store     sp, [r0+offsSP] ; save SP

  ; Restore the new context
  load      r0, newRunning
  load      sp, [r0+offsSP] ; restore SP
  pop       r31 ; restore regs
  ...
  pop       r0
  pop       spsw
  pop       ssp
  pop       spc
  }
}
```

3. (10 poena)

```
#include <fcntl.h>
  #include <sys/stat.h>
  #include <semaphore.h>

const size_t BBSIZE = ...;

static sem_t *mutex, *space_available, *item_available;

struct bbuffer {
  char buf[BBSIZE];
  int head, tail;
};


int  bbuf_init (struct bbuffer* bb) {
  mutex = sem_open("/bounded_buf_mx",O_CREAT,O_RDWR,1);
  space_available = sem_open("/bounded_buf_sa",O_CREAT,O_RDWR,BBSIZE);
  item_available = sem_open("/bounded_buf_ia",O_CREAT,O_RDWR,0);
  if (!mutex || !space_available || !item_available) {
    if (mutex) sem_close(mutex);
    if (space_available) sem_close(space_available);
    if (item_available) sem_close(item_available);
    return -1;
  }
  return 0;
}


void  bbuf_close (struct bbuffer* bb) {
  if (mutex) sem_close(mutex);
  if (space_available) sem_close(space_available);
  if (item_available) sem_close(item_available);
}


void  bbuf_append (struct bbuffer* bb, char c) {
  sem_wait(space_available);
  sem_wait(mutex);
  bb->buf[bb->tail] = c;
  bb->tail = (bb->tail+1)%BBSIZE;
  sem_post(mutex);
  sem_post(item_available);
}


char bbuf_take (struct bbuffer* bb) {
  sem_wait(item_available);
  sem_wait(mutex);
  char c = bb->buf[bb->head];
  bb->head = (bb->head+1)%BBSIZE;
  sem_post(mutex);
  sem_post(space_available);
  return c;
}
```