Rešenja kolokvijuma iz Operativnih sistema 2, januar 2021.

1. (10 poena)

```
class Scheduler {
public:
  Scheduler () {}
  PCB* get ();
  void put (PCB*, bool wasBlocked);

private:
  class ProcList {
  public:
    ProcList () : head(0), tail(0) {}
    void put (PCB* p);
    PCB* get ();
  private:
    PCB *head, *tail;
  };

  ProcList ready[MaxPri+1];
};

inline void Scheduler::ProcList::put (PCB* p) {
  if (tail) tail->next = p;
  else head = tail = p;
  p->next = 0;
}

inline PCB* Scheduler::ProcList::get () {
  PCB* ret = head;
  if (head) head = head->next;
  if (!head) tail = 0;
  return ret;
}

PCB* Scheduler:: put (PCB* p, bool wasBlocked) {
  if (pcb==0) return; // Exception!
  if (wasBlocked) {
    if ((pcb->curPri > 0) &&
        (pcb->curPri+PriMargin > pcb->defPri)) pcb->curPri--;
  } else {
    if ((pcb->curPri < MaxPri) &&
        (pcb->curPri < pcb->defPri+PriMargin)) pcb->curPri++;
  }
  ready[pcb->curPri].put(p);
}

PCB* Scheduler::get () {
  for (int i=0; i<=MaxPri; i++) {
    PCB* p = ready[i].get();
    if (p) return p;
  }
  return 0;
}
```

2. (10 poena) Postoji sledeća (sigurna) sekvenca kojim procesi mogu da dobiju zahtevane resurse i izvrše se do kraja, pa sistem nije u mrtvoj blokadi: *P*3, *P*4, *P*2, *P*1.

Allocation

	A	*B*	*C*
*P*1	1	0	1
*P*2	0	1	1
*P*3	2	1	1
*P*4	1	0	0

Request

	A	*B*	*C*
*P*1	0	1	3
*P*2	3	0	1
*P*3	0	0	0
*P*4	2	1	0

Available

A	*B*	*C*
0	1	1

Allocation

	A	*B*	*C*
*P*1	1	0	1
*P*2	0	1	1
~~*P*3~~	~~2~~	~~1~~	~~1~~
*P*4	1	0	0

Request

	A	*B*	*C*
*P*1	0	1	3
*P*2	3	0	1
~~*P*3~~	~~0~~	~~0~~	~~0~~
*P*4	2	1	0

Available

A	*B*	*C*
2	2	2

Allocation

	A	*B*	*C*
*P*1	1	0	1
*P*2	0	1	1
~~*P*3~~	~~2~~	~~1~~	~~1~~
~~*P*4~~	~~1~~	~~0~~	~~0~~

Request

	A	*B*	*C*
*P*1	0	1	3
*P*2	3	0	1
~~*P*3~~	~~0~~	~~0~~	~~0~~
~~*P*4~~	~~2~~	~~1~~	~~0~~

Available

A	*B*	*C*
3	2	2

Allocation

	A	*B*	*C*
*P*1	1	0	1
~~*P*2~~	~~0~~	~~1~~	~~1~~
~~*P*3~~	~~2~~	~~1~~	~~1~~
~~*P*4~~	~~1~~	~~0~~	~~0~~

Request

	A	*B*	*C*
*P*1	0	1	3
~~*P*2~~	~~3~~	~~0~~	~~1~~
~~*P*3~~	~~0~~	~~0~~	~~0~~
~~*P*4~~	~~2~~	~~1~~	~~0~~

Available

A	*B*	*C*
3	3	3

Allocation

	A	*B*	*C*
~~*P*1~~	~~1~~	~~0~~	~~1~~
~~*P*2~~	~~0~~	~~1~~	~~1~~
~~*P*3~~	~~2~~	~~1~~	~~1~~
*P*4	1	0	0

Request

	A	*B*	*C*
~~*P*1~~	~~0~~	~~1~~	~~3~~
~~*P*2~~	~~3~~	~~0~~	~~1~~
~~*P*3~~	~~0~~	~~0~~	~~0~~
*P*4	2	1	0

Available

A	*B*	*C*
4	3	4

3. (10 poena)

a)(5)

```
inline void Buddy::split (char* seg, int upper, int lower) {
  while (--upper>=lower)
    bucket[upper].put(seg + (1<<upper)*BLOCK_SIZE);
}
```

b)(5)

```
void* Buddy::alloc (int size) {
  if (size<0 || size>=BUCKET_SIZE) return 0; // Exception

  for (int current=size; current<BUCKET_SIZE; current++) {
    char* p = bucket[current].get();
    if (!p) continue;
    split(p,current,size);
    return p;
  }
  return 0;
}
```

4. (10 poena)

```
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

struct input_msg {
    int pid;
    char msg[50];
};

#define N 100
#define STOP "STOP"


int makeHash(char *msg);
void getMsg(char *msg);

// First program
int main()
{
    mkfifo("input", 0666);
    int fd = open("input", O_RDONLY);

    while (1) {
        struct input_msg msg;

        if (read(fd, &msg, sizeof(msg)) <= 0) {
            continue;
        }

        if (!strcmp(msg.msg, STOP)) {
            break;
        }

        char outName[20];
        sprintf(outName, "pipe%d", msg.pid);
        int outFd = open(outName, O_WRONLY);
        int result = makeHash(msg.msg);
        result = msg.pid;
        write(outFd, &result, sizeof(result));
        close(outFd);
    }

    close(fd);

    return 0;
}

// Second program
int main(int argnum, char **args) {
    int pid;
    sscanf(args[1], "%d", &pid);

    char outName[20];
    sprintf(outName, "pipe%d", pid);
    mkfifo(outName, 0666);

    int fd = open("input", O_WRONLY);
    struct input_msg msg;

    msg.pid = pid;

    if (pid == N) {
        strcpy(msg.msg, STOP);
    } else {
        getMsg(msg.msg);
    }

    write(fd, &msg, sizeof(msg));
    close(fd);
    if (pid == N) {
      return 0;
    }

    int outFd = open(outName, O_RDONLY);
    int result;
    read(outFd, &result, sizeof(result));
    printf("Result=%d\n", pid, result);
    close(outFd);


    return 0;
}
```