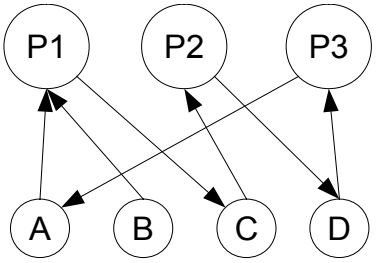
Rešenja drugog kolokvijuma iz Operativnih sistema 2, januar 2020.

1. (10 poena) a)(7) Ideja algoritma jeste u tome da se ispita da li postoji ciklična kombinacija držanja i čekanja ugnežđenih kritičnih sekcija različitih procesa koja je u skladu sa strukturom njihovog korišćenja. Drugim rečima, konstruiše se graf alokacije resursa (resursi su ovde kritične sekcije) za sve situacije u kojima procesi mogu da drže neke resurse i čekaju na neke resurse, u skladu sa strukturom programa, i proverava se da li u takvim grafovima postoje petlje. Mrtva blokada je moguća ako i samo ako se otkrije takva petlja.
Preciznije, algoritam izgleda ovako. Posmatraju se sve staze od korena svakog stabla ugnežđivanja kritičnih sekcija do svakog od listova tog stabla. Za svaku takvu stazu posmatraju se podstaze od korena, a koje sadrže najmanje dva čvora sa resursima. Za svaku od odabranih podstaza pretpostavlja se da je odgovarajući proces zauzeo sve resurse osim poslednjeg u stazi, a čeka na taj poslednji (najdalji od korena). Posmatraju se sve kombinacije takvih podstaza, po jedna iz svakog stabla (tj. po jedna za svaki proces). Za svaku takvu kombinaciju konsturiše se graf zauzeća resursa od strane procesa: čvorovi su procesi i resursi (ovde kritične sekcije), grana zauzeća polazi od resursa i ide do procesa koji je taj resurs zauzeo (za sve čvorove u podstazi osim poslednjeg), a grana potražnje ide od procesa do resursa na koji on čeka (za poslednji resurs u podstazi). Pritom se isključuju (ne uzimaju u obzir) oni grafovi zauzeća koji ne zadovoljavaju oba sledeća dva uslova:
1) Iz svakog čvora može polaziti najviše jedna izlazna grana (svaki resurs može držati najviše jedan proces; svaki proces može čekati najviše na jedan resurs, što je zadovoljeno samom kontrukcijom grafa).
2) Ako u čvor za resurs ulazi grana, onda iz nje mora izlaziti grana (ako neki proces čeka na neki resurs, onda je taj resurs zauzet).
Za svaki tako konstruisan graf zauzeća proverava se postojanje petlje.
Ovaj algoritam se može optimizovati smanjivanjem prostora pretrage, recimo ovako: ako u datom grafu zauzeća nije zadovoljen uslov 1), onda taj uslov neće biti zadovoljen ni za grafove za sve druge kombinacije podstaza u kom su te podstaze nadskupovi podstaza prvog grafa, tj. pretraga ne treba dalje da analizira kombinacije dužih nadstaza.

b)(3) Za datu šumu stabala, podstaze koje se uzimaju u obzir su sledeće:

P1: {A-B, A-B-C}; P2: {C-D, C-E}; P3: {D-A, D-A-E}

Za kombinaciju staza: P1: A-B-C, P2: C-D, P3: D-A graf zauzeća izgleda ovako i poseduje



petlju A-P1-C-P2-D-P3-A, pa je mrtva blokada moguća:

2. (10 poena)

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>

inline void handle_err (const char* msg) {
  fprintf(stderr, "Error: %s", msg); exit(-1);
}

const size_t MAX_ARRAY_SIZE = ...;
const size_t MEM_SIZE = (MAX_ARRAY_SIZE+2)*sizeof(unsigned);

int main (int argc, char* argv[]) {
  if (argc != 2) handle_err("Expecting one argument for the file.");

  fd = open(argv[1], O_RDWR);
  if (fd == -1) handle_err("Cannot open file.");

  void* addr = mmap(NULL, MEM_SIZE, PROT_WRITE, MAP_SHARED, fd, 0);
  if (addr == MAP_FAILED) handle_err("Cannot map file.");

  unsigned size = *((unsigned*)addr+1);
  unsigned *array = (unsigned*)addr+2;
  unsigned max = 0;
  for (unsigned i=0; i<size; i++)
    if (array[i] > max) max = i;
  *(unsigned*)addr = max;

  munmap(addr, MEM_SIZE);
  close(fd);
  exit(0);
}
```

3. (10 poena)

a)(5)

n	*Početne adrese (hex) slobodnih blokova veličine* 2^n
4	-
3	-
2	AC 000
1	AA 000
0	A3 000

b)(5)

n	*Početne adrese (hex) slobodnih blokova veličine* 2^n
4	-
3	A8 000
2	-
1	A2 000
0	-