Rešenja drugog kolokvijuma iz Operativnih sistema 2, januar 2026.

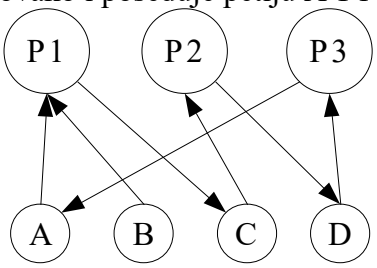
1. (10 poena) Za objašnjenje oznaka i algoritma koji analizira sve mogućnosti i pronalazi situacije sa mrtvom blokadom pogledati rešenje zadatka sa Drugog kolokvijuma iz januara 2020. godine. Za datu šumu stabala, podstaze koje se uzimaju u obzir su sledeće:

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

Sve njihove kombinacije i značenje tih kombinacija dati su u sledećoj tabeli.

Kombinacija staza zauzeća resursa	*Ishod*
P1: A-B; P2: C-D; P3: D-A	Nema mrtve blokade
P1: A-B; P2: C-D; P3: D-A-E	Nemoguća (A zauzela dva procesa)
P1: A-B; P2: C-E; P3: D-A	Nema mrtve blokade
P1: A-B; P2: C-E; P3: D-A-E	Nemoguća (A zauzela dva procesa)
P1: A-B-C; P2: C-D; P3: D-A	Mrtva blokada
P1: A-B-C; P2: C-D; P3: D-A-E	Nemoguća (A zauzela dva procesa)
P1: A-B-C; P2: C-E; P3: D-A	Nema mrtve blokade
P1: A-B-C; P2: C-E; P3: D-A-E	Nemoguća (A zauzela dva procesa)

Prema tome, jedina kombinacija staza koja dovodi do mrtve blokade je: P1: A-B-C, P2: C-D, P3: D-A, a graf zauzeća izgleda ovako i poseduje petlju A-P1-C-P2-D-P3-A:



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);
}

struct Node {
  unsigned val;
  size_t lchild, rchild;
};

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

bool find (Node arr[], unsigned key) {
  Node* node = arr;
  while (node) {
    if (node->val==key) return true;
    if (node->val>key) node = node->lchild ? &arr[node->lchild] : nullptr;
    if (node->val<key) node = node->rchild ? &arr[node->rchild] : nullptr;
  }
  return false;
}

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

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

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

  unsigned key = *(unsigned*)addr;
  Node* root = (Node*)((unsigned*)addr+1);

  bool found = find(root,key);
  printf(found?"Found.\n":"Not found.\n");

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

3. (10 poena)

```
#include <stdint.h>
const int N = 6;
struct buddy_t {
  uint64_t array_size;
  int64_t buddy[N];
};
const uint64_t PAGE_SIZE_WIDTH = 12;
const uint64_t VA_SIZE_WIDTH = 64;
const uint64_t ONE = 1;
```

```
const uint64_t START_PAGE = (ONE << VA_SIZE_WIDTH - PAGE_SIZE_WIDTH) - (ONE
<< N - 1);

void* get_page_address(struct buddy_t buddy, uint64_t page_num) {
    if (buddy.array_size == 0) {
        return 0;
    }
    uint64_t max_page = 1ULL << buddy.array_size - 1;
    if (page_num < 0 || page_num >= max_page) {
        return 0;
    }

    uint64_t address = (START_PAGE + page_num) << PAGE_SIZE_WIDTH;
    return (void*)address;
}
int increase_memory(struct buddy_t buddy) {
    if (buddy.array_size >= N) {
        return -1;
    }

    if (buddy.array_size > 0 && buddy.buddy[buddy.array_size - 1] != -1) {
        // Whole space is free, just double the size by merging
        buddy.buddy[buddy.array_size] = buddy.buddy[buddy.array_size - 1];
        buddy.buddy[buddy.array_size - 1] = -1;
        buddy.array_size += 1;
    } else {
        // Buddy is not free, so merging is not possible, just add a new
space
        const int64_t SIGNED_ONE = 1;
        int64_t new_block_start = buddy.array_size == 0 ? 0 : SIGNED_ONE <<
buddy.array_size - 1;
        buddy.buddy[buddy.array_size] = new_block_start;
        buddy.array_size += 1;
        int64_t *block_ptr = get_page_address(buddy, new_block_start);
        if (block_ptr == 0) {
            return -2;
        }
        *block_ptr = -1;
    }

    return 0;
}
```