Rešenja zadataka za kolokvijum iz Operativnih sistema 1 - Rok 4/2025.

1. (10 poena)

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>

#include "bbuffer.h"

typedef BoundedBuffer<const char, 32> Buffer;

#define handle_error(msg) \
  do { perror(msg); exit(EXIT_FAILURE); } while (0)

int main() {
  void* shmem = mmap(NULL, sizeof(Buffer), PROT_READ|PROT_WRITE,
                     MAP_SHARED|MAP_ANONYMOUS, -1, 0);
  if (shmem == MAP_FAILED) handle_error("mmap failed");

  Buffer* buffer = new (shmem) Buffer;

  pid_t pid = fork();
  if (pid < 0) {
    hande_error("fork failed");
  } else if (pid == 0) {
      static const char str[] = "Hello world!";
      for (int i=0; i<=strlen(str); i++)
        buffer->append(str[i]);
  } else {
      do {
        const char c = buffer->take();
        putchar(c);
      } while (c);
      wait(NULL);
      buffer->~Buffer();
  }

  munmap(shmem, size);
  return 0;
}
```

2. (10 poena)

```
class Event {
public:
  Event (int init=0) : val(init?1:0) {}

  void wait();
  void signal();

private:
  int val;
  ThreadQueue blocked;
};
```

```
void Event::wait () {
  lock();
  Thread* oldRunning = Thread::runningThread;
  if (--val<0)
    this->blocked.put(oldRunning);
  else
    Scheduler::put(oldRunning);
  Thread* newRunning = Thread::runningThread = Scheduler::get();
  if (oldRunning!=newRunning)
    Thread::yield(oldRunning,newRunning);
  unlock();
}
```

```
void Semaphore::signal () {
  lock();
  Thread* oldRunning = Thread::runningThread;
  if (++val<=0)
    Scheduler::put(this->blocked.get());
  else
    val = 1;
  Scheduler::put(oldRunning);
  Thread* newRunning = Thread::runningThread = Scheduler::get();
  if (oldRunning!=newRunning)
    Thread::yield(oldRunning,newRunning);
  unlock();
}
```

3. (10 poena)

```
Byte* BlockIOCache::getBlock (BlkNo blk) {
  // Find the requested block in the cache and return it if present:
  int entry = hash(blk);
  for (int i=0; i<map[entry]; i++)
    if (entries[entry][i].blkNo==blk) {
      entries[entry][i].refCounter++;
      return entries[entry][i].buf;
    }
  // The block is not in the cache, find a free slot to load it:
  int free = map[entry];
  if (free==CASHSIZE) free = evict(entry);
  if (free==CASHSIZE) return 0;
  // Load the requested block:
  entries[entry][free].blkNo = blk;
  entries[entry][free].refCounter = 1;
  map[entry].count++;
  ioRead(dev,blk,entries[entry][free].buf);
  return entries[free].buf;
}
```

4. (10 poena)

```
#include <unistd.h>

ListNode* getNode (int fd, long index) {
  static ListNode buffer;
  static long loadedNode = -1;

  if (loadedNode!=index) {
    if (read(fd,&buffer,sizeof(ListNode))<sizeof(ListNode))
      handleError("Corrupted file");
    loadedNode = index;
  }
  return &buffer;
}
```

```
long getValue (int fd, long key) {
  long index = 0;
  do {
    ListNode* node = getNode(index);
    for (short i=0; i<node->count; i++)
      if (node->keys[i]==key)
        return node->values[i];  // Key found
    index = node->next;
  } while (index);
  return -1;  // Key not found
```