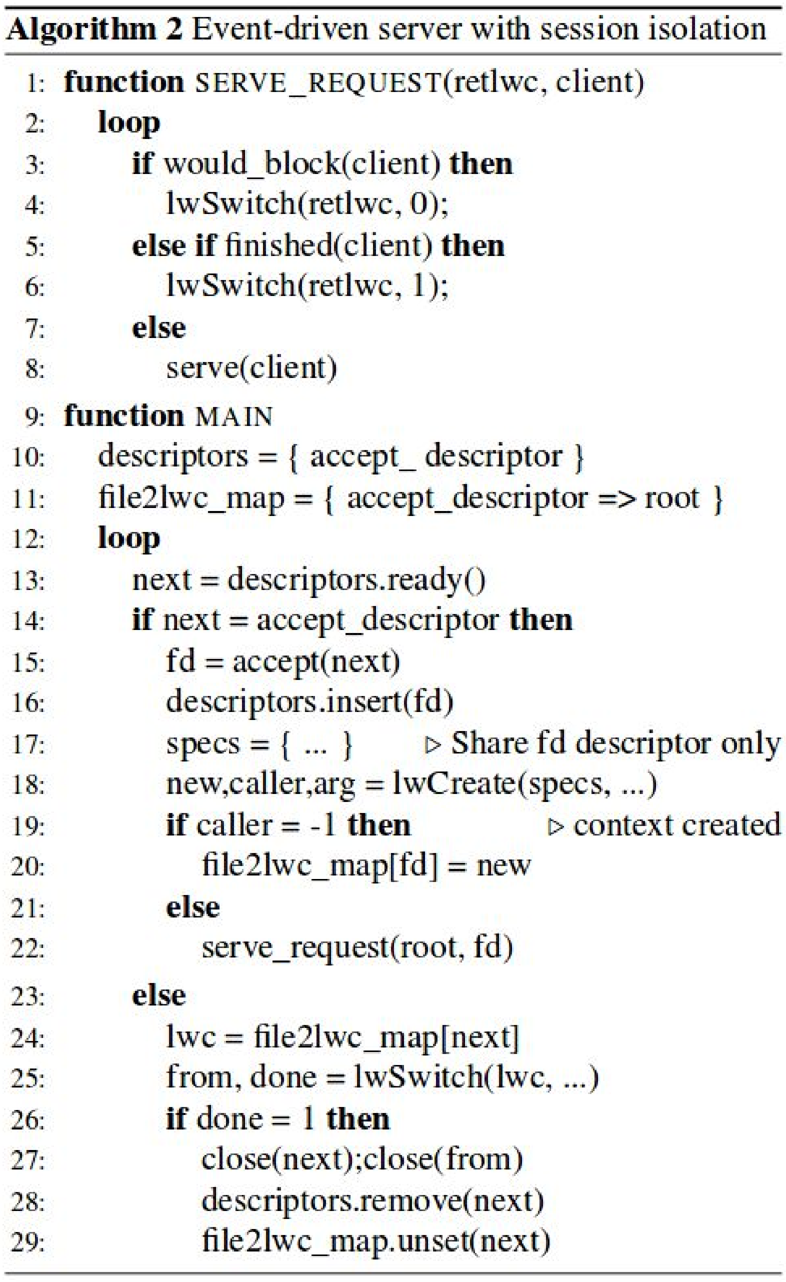
CSL862 Midterm Exam
Advanced Topics in Operating Systems
Sem 1 2017-18
18 September 2017

Answer all 6 questions Max. Marks: 25

1. On the duality of OS structures: What is the dual of condition variable “wait()” operation in a message-oriented system? What is the dual of condition variable “signal()” operation in a message-oriented system? What are the semantics of this “dual of condition variables” in the message-oriented system: Hoare or Mesa? Explain briefly. [5]

2. Light-weight contexts : Consider the following code (taken from the paper) for an event-driven server with session isolation.



Algorithm 2 Event-driven server with session isolation

```
 1: function SERVE_REQUEST(retlwc, client)
 2:     loop
 3:         if would_block(client) then
 4:             lwSwitch(retlwc, 0);
 5:         else if finished(client) then
 6:             lwSwitch(retlwc, 1);
 7:         else
 8:             serve(client)
 9: function MAIN
10:     descriptors = { accept_ descriptor }
11:     file2lwc_map = { accept_descriptor => root }
12:     loop
13:         next = descriptors.ready()
14:         if next = accept_descriptor then
15:             fd = accept(next)
16:             descriptors.insert(fd)
17:             specs = { ... }          ▷ Share fd descriptor only
18:             new,caller,arg = lwCreate(specs, ...)
19:             if caller = -1 then              ▷ context created
20:                 file2lwc_map[fd] = new
21:             else
22:                 serve_request(root, fd)
23:         else
24:             lwc = file2lwc_map[next]
25:             from, done = lwSwitch(lwc, ...)
26:             if done = 1 then
27:                 close(next);close(from)
28:                 descriptors.remove(next)
29:                 file2lwc_map.unset(next)
```

Assume that there are two clients and after the server is started, they both initiate a connection with the server. These clients keep making requests to the server alternately and never finish. I.e., the request pattern to the server from the clients is:
Client1Initiate → Client2Initiate → Client1Request → Client2Request → Client1Request → Client2Request → Client1Request → Client2Request → …

Show the exact sequence of execution of the statements in the code above for this pattern of client requests. For each statement, mention the lwC that executes that statement. [4]

3. Huge page management :

a. Why do some applications speed up by using huge pages? Why do some applications slow down by using huge pages? State the nature of application that would result in that behaviour and briefly explain the reason. [2]

b. Fragmentation

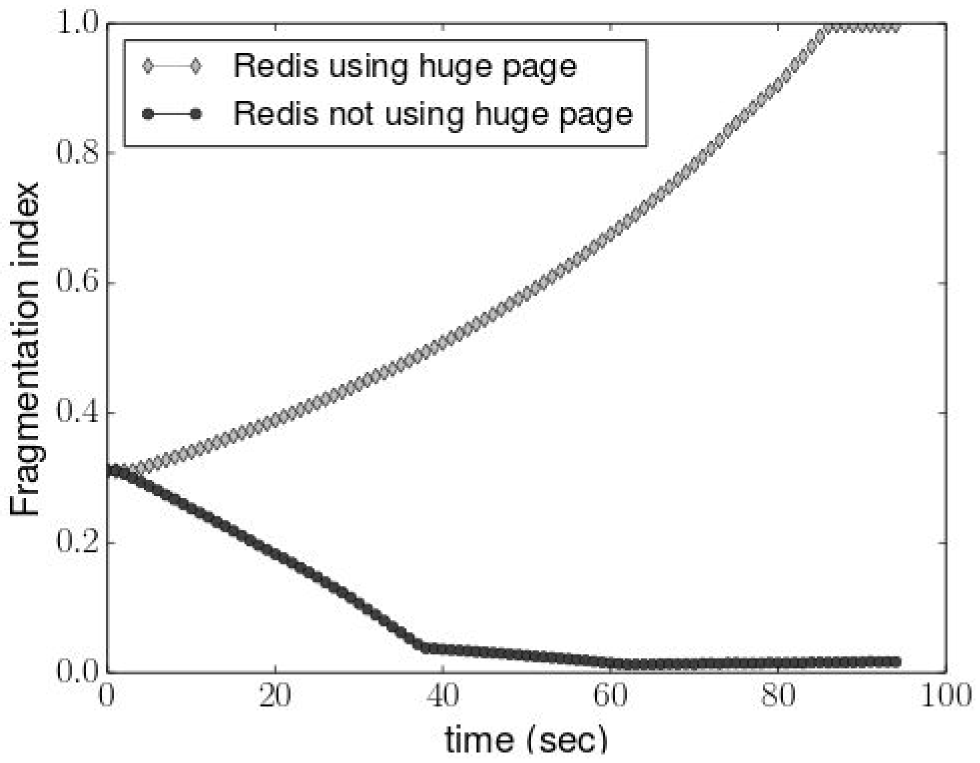


Figure 1: Fragmentation index in Linux when running a Redis server, with Linux using (and not using) huge pages. The System has 24 GB memory. Redis uses 13 GB, other processes use 5 GB, and system has 6 GB free memory.

Why does fragmentation increase for Redis when using huge pages? [1.5]
Why does fragmentation decrease when using regular pages? [1.5]

4. Crash Refinement:

Consider the following specification of the mknod operation:

```
def mknod(self, parent, name, mtime, mode):
  # Name must not exist in parent.
  if self._childmap[(parent, name)] > 0:
    return -errno.EEXIST

  # The new ino must be valid & not already exist.
  ino = InoT()
  assertion(ino > 0)
  assertion(Not(self._parentmap[ino] > 0))

  with self.transaction():
    # Update the directory structure.
    self._childmap[(parent, name)] = ino
    self._parentmap[ino] = parent
    # Initialize inode metadata.
    self._mtimemap[ino] = mtime
    self._modemap[ino]  = mode
    self._sizemap[ino]  = 0

  return ino
```

Now consider the following definition of crash refinement without recovery:

$$\forall s_0, s_1, \boldsymbol{x}, \boldsymbol{b}_1.\ \exists \boldsymbol{b}_0.\ \ (s_0 \sim_{\mathcal{I}_0,\mathcal{I}_1} s_1) \Rightarrow (s_0' \sim_{\mathcal{I}_0,\mathcal{I}_1} s_1')$$
$$\text{where } s_0' = f_0(s_0, \boldsymbol{x}, \boldsymbol{b}_0) \text{ and } s_1' = f_1(s_1, \boldsymbol{x}, \boldsymbol{b}_1).$$

Explain what is s0, s1, I0, I1, x, b0, b1. Why is there an existential quantifier (exists) over b0 while there is a universal quantifier (for-all) over s0, s1, x, and b1? What would be potential values of b0 for the specification of mknod given above? [4]

5. CertiKOS

The paper mentions that one of the limitations of the current verification approach is that it assumes strong sequential consistency for all atomic instructions. What is sequential consistency, and where is this assumption used?   [2]

Verification usually involves a layered approach, where-in properties of the lower layer are proved in isolation, and then those properties are used in the higher layers. Give one example of such a layered approach in CertiKOS. [2]

6. RPCs over RDMA

a. How are coroutines used to improve RPC throughput? Why coroutines? [1]

b. Why is the the “validate” phase necessary in FaSST’s transaction protocol? [1.5]

c. Why is the the “validate” phase *sufficient* in FaSST’s transaction protocol? I.e., what property does it guarantee? [1.5]