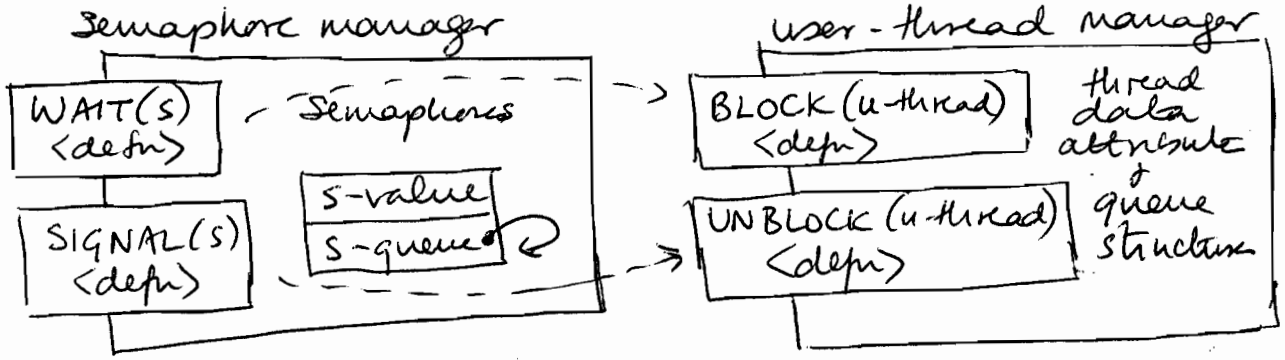
1999 Concurrent Systems - solution. Paper 3 p3q6 JMB

a)



6

All u-threads are multiplexed on one OS process.

WAIT(s): add calling thread to s-queue on need to block then call BLOCK(u-thread) + schedule

SIGNAL(s): if s has a queue, remove one thread and call UNBLOCK(u-thread).

b) (i) thread management is in the OS.
the OS schedules threads.
the RTS is involved only in create/kill(thread) - making threads known to the OS.
eg k-thread-id := create(thread) system call
a blocking call to WAIT(s): manage s-queue as before and call OS block(k-thread-id).
an unblocking call to SIGNAL(s): manage s-queue as before and call OS unblock(k-thread-id).

5

Assumption - no limit to # of threads - can use one lc-thread for each u-thread.

(ii) OS guarantees that some fixed number of k-threads per address space can RUN.
Assumption: OS tells RTS if a thread makes a blocking system call + passes its context. The RTS passes the context of the runnable thread to be run in its place. Blocking at user level on WAIT(s) requires s-data value + queue management + OS calls to block this thread + set up another. An unblocking call to SIGNAL(s): manage s-queue + change state of unblocked thread at user level

5

c) Semaphore data structures are shared + read/write not atomic. WAIT + SIGNAL must be made atomic - one (monitor) lock/module or a lock (boolean) per semaphore. Threads spin (for a short time) on these locks on which they are waiting to block

4

(20)