Advanced Topics in OS: Multithreading 2

Michael Elhadad

January 22, 1998

1. Non-blocking IO and System Calls

1. System calls that blocks block the whole process (pod) and not only the thread that initiated the call. Can be problematic.

2. Solution is to link with a special Non-blocking IO library. Because there is no kernel support for threads at present, system calls effectively block the entire pod. By linking in the non-blocking I/O library (-lnbio) ahead of the LWP library, you can alleviate this problem for those system calls that can issue a signal when a system call would be profitable to try. This library (which redefines some system calls) uses asynchronous I/O and events (for example, SIGCHLD and SIGIO) to make blocking less painful. The system calls remapped by the nbio library are: open(2V), socket(2), pipe(2V), close(2V), read(2V), write(2V), send(2), recv(2), accept(2), connect(2), select (2) and wait(2V).

Example 1: Create a thread to read and one to compute.

```
/* Compile with:
 * gcc reader.c -o reader -lnbio -llwp
 * This demonstrates the importance using the non-blocking IO library
 */

#include <lwp/lwp.h>
#include <lwp/stackdep.h>
#define MAXPRIO 10

int reader();
void otherwork();

void main(int arg, char** argv) {
  thread_t tid;
  pod_setmaxpri(MAXPRIO);
  lwp_setstkcache(3000, 2);
  lwp_create(&tid, reader, MAXPRIO, 0, lwp_newstk(), 0);
  lwp_setpri(SELF, MINPRIO);
  otherwork();
  exit(0);
}

int reader() {
  char buf[256];
  int cnt;
  for (;;) {
    cnt = read(0, buf, 256);
    buf[cnt] = 0;
```

```
    printf("\nGot %s\n", buf);
  }
}

void otherwork()
{
  struct timeval time;
  time.tv_sec = 2;
  time.tv_usec = 0;
  for (;;) {
    printf("Otherwork...\n");
    lwp_sleep(&time);
  }
}
```

In Example 2, the server waits for connections on a datagram socket and forks a thread to serve each connection. Each forked thread continues to work as the server continues to wait for connections.

Datagram socket primitives are: recvfrom - server receives a single packet from a client and sendto - client sends a single packet. In datagram mode, there is no connection (no stream) and therefore no need for listen/accept. The recvfrom call waits for a single packet (like accept would wait in stream mode). On the client side, there is no need for connect as the sendto call makes the whole work.

Example 2: Sleep Server and Sleep Client

```
/* Sleep server program
 * Compile with:
 * gcc sleep-server.c -o sleep-server -lnbio -llwp
 * This demonstrates the importance using the non-blocking IO library
 */

#include <lwp/lwp.h>
#include <lwp/stackdep.h>
#include <lwp/lwperror.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <errno.h>

#define MYPORT 8889
#define BUFSIZE 10
#define MAXPRIO 10

struct message {
  int timeout;
  int msgsize;
  char buf[BUFSIZE];
} message;

extern int errno;

void compute(int timeout, char* msg);

void main(int arg, char** argv) {
  int s;
  struct sockaddr_in addr;
```

```
  int len = sizeof(struct sockaddr_in);
  int fromlen;
  int rlen;
  thread_t tid;

  pod_setmaxpri(MAXPRIO);
  lwp_setstkcache(5000, 5);
  if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
    perror("Can't get socket");
    exit(1);
  }
  addr.sin_addr.s_addr = INADDR_ANY;   /* accept connection from anybody */
  addr.sin_family = AF_INET;
  addr.sin_port = MYPORT;
  if (bind(s, (struct sockaddr_in *)&addr, len) < 0) {
    perror("bind");
    close(s);
    exit(1);
  }
  if (getsockname(s, (caddr_t)&addr, &len) != 0) {
    perror("Can't get name");
    close(s);
    exit(1);
  }

  /* Server loop */
  for (;;) {
    do {
      fromlen = len;
      rlen = recvfrom(s, (caddr_t)&message, sizeof(struct message), 0,
      &addr, &fromlen);
    } while ((rlen == -1) && (errno == EINTR));
    if (rlen == -1) {
      perror("recvfrom");
      exit(1);
    }
    lwp_create(&tid, compute, MINPRIO, 0, lwp_newstk(),
               2, message.timeout, message.buf);
  }
  exit(0);
}

void compute(int timeout, char* msg)
{
  struct timeval time;
  time.tv_sec = timeout;
  time.tv_usec = 0;
  lwp_sleep(&time);
  printf("%s slept %d secs\n", msg, timeout);
}
```

Example 2: Sleep client.

```
/* sleep client - to use with sleep-server              */
/* Compile with: gcc slp -o slp                         */
/* Usage: slp <servername> <timeout in seconds> <message> */
```

```
/* message length limited to 9 characters                  */

#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>

#define MYPORT 8889
#define BUFSIZE 10
struct message {
  int timeout;
  int msgsize;
  char buf[BUFSIZE];
} message;
extern int errno;

void main(int argc, char** argv) {
  int s;
  struct sockaddr_in addr;
  int len = sizeof(struct sockaddr_in);
  int err;
  struct hostent *hp;
  char* server;

  if (argc != 4) {
    printf("usage: %s server seconds message\n", argv[0]);
    exit(2);
  }
  server = argv[1];
  message.timeout = atoi(argv[2]);
  message.msgsize = strlen(argv[3]) + 1;
  bcopy(argv[3], message.buf, message.msgsize);

  if ((hp = gethostbyname(server)) == 0) {
    printf("can't get host name\n");
    exit(1);
  }
  bcopy(hp->h_addr, &addr.sin_addr, hp->h_length);
  addr.sin_family = AF_INET;
  addr.sin_port = MYPORT;

  if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
    perror("can't get socket");
    exit(1);
  }

  do {
    err = sendto(s, (caddr_t)&message, sizeof(message), 0, &addr, len);
  } while ((err == -1) && (errno == EINTR));
  if (err == -1) {
    perror("sendto");
    exit(1);
  }
  exit(0);
}
```

The final example shows how to fork children and wait for them in a separate thread while regular work continues. The system call `wait` blocks a single thread and not the whole pod.

Example 3: Fork children and wait for them in a separate thread.

```
/* Non-blocking IO / Thread example
 * Compile as: gcc wait-thread.c -o wait-thread -lnbio -llwp
 * Fork 2 children and wait for them (with wait) without blocking
 */
#include <lwp/lwp.h>
#include <lwp/lwpmachdep.h>
#include <signal.h>

void main(int argc, char** argv)
{
  int child;
  union wait stat;
  void otherwork();

  pod_setmaxpri(10);
  lwp_setstkcache(1000, 2);
  lwp_create(0, otherwork, MINPRIO, 0, lwp_newstk(), 0);
  if (fork() == 0) {
    sleep(5);
    exit(7);
  } else if (fork() == 0) {
    sleep(3);
    exit(5);
  }
  /* Reap children */
  for (;;) {
    child = wait(&stat);
    printf("%d got %d\n", child, stat.w_retcode);
    if (child == -1) {
      perror("wait");
      break;
    }
  }
  exit(0);
}

void otherwork()
{
  struct timeval time;
  time.tv_sec = 2;
  time.tv_usec = 0;
  for (;;) {
    printf("Otherwork...\n");
    lwp_sleep(&time);
  }
}
```

Example 3: Output: note that otherwork() continues working

```
Otherwork...
```

```
Otherwork...
Otherwork...
4794 got 0
Otherwork...
4793 got 0
-1 got 0
wait: No children
Otherwork...
Otherwork...
```

2. Monitors and Conditions

1. Second mode of inter-thread communication: monitors (as opposed to rendez-vous).
2. Implements critical sections with a model similar to sleep()/wakeup() or to semaphores. Use here name of *condition variables*.
3. Use monitors as a synchronization tool and agents to handle interrupts - no need to disable interrupts (and therefore less risk to drop interrupts).
4. A condition variable has a state associated with it. For example, for the consumer/producer example, the state can be *empty* and one *full*. Because the buffer state can be accessed by several threads, it is protected by a monitor. Otherwise, a thread could decide to wait for a state change, only to have the state change before the wait can be executed, resulting in deadlock. Therefore, both the waiter and the notifier must access the state in a monitor. Typical code looks like:

   ```
   mon_enter(m);
   ...
   while (!state)
     cv_wait(cv);
   ...
   mon_exit(cv);
   ```

   The while loop is necessary because if there are several threads waiting in the monitor when the condition is broadcast, all of them wake up, but the first thread to gain entry to the monitor may alter the state, invalidating it for the other awakened threads.

   For example, if 2 producers are awakened because the buffer is no longer full, the first one may fill the buffer again and wait, leaving the second one to run. The second producer must not add to the buffer now, because it is full again.

5. Condition variables are manipulated through the primitives:

   - cv_broadcast(): awakens all threads blocked on a condition
   - cv_notify(): awakens a single thread blocked on a condition. This can result in a deadlock if the awakened thread is not the one that should be notified.
   - cv_wait(): wait on a condition.
   - cv_waiter(): which threads are waiting on the condition.

   Condition variables are bound to a single monitor.

Example 4: Consumer / Producer with monitors.

```
/* Monitor example: producer/consumer with scheduling
 * Compile: gcc producer.c -o producer -llwp
 */
#include <lwp/lwp.h>
#include <lwp/stackdep.h>

void producer();
void consumer();
void sch();
thread_t c1, c2, sched;
mon_t m1;                      /* Monitor              */
cv_t notempty, notfull;        /* Condition variables */
int cnt = 0;                   /* how many chars are in buffer    */
int in = 0;                    /* Where to add new char in buffer */
int out = 0;                   /* Where to read char from buffer  */
#define MAXBUF 20
char buf[MAXBUF];
#define MAXPRIO 10

int main(int argc, char** argv)
{
  pod_setmaxpri(MAXPRIO);
  mon_create(&m1);
  cv_create(&notempty, m1);
  cv_create(&notfull, m1);
  lwp_setstkcache(3000, 3);
  lwp_create(&sched, sch,   MAXPRIO, 0, lwp_newstk(), 0);
  lwp_create(&c1, producer, MINPRIO, 0, lwp_newstk(), 0);
  lwp_create(&c2, consumer, MINPRIO+1, 0, lwp_newstk(), 0);
  exit(0);
}

/* Add a character to the buffer */
void bufput(char c)
{
  MONITOR(m1);    /* acquire monitor m1 and release it on exit */
  /* Buffer never gets > MAXBUF */
  while (cnt == MAXBUF) {
    printf("waiting on notfull\n");
    cv_wait(notfull);
  }
  printf("Putting %c into %d\n", c, in);
  buf[in] = c;
  in = (in + 1) % MAXBUF;
  cnt++;
  cv_broadcast(notempty);
  printf("Broadcast done\n");
}

void bufget(char* c)
{
  MONITOR(m1);
  /* buffer never gets < 0 */
  while (cnt == 0) {
    printf("waiting on nonempty\n");
```

```
    cv_wait(notempty);
  }
  *c = buf[out];
  out = (out + 1) % MAXBUF;
  cnt--;
  cv_broadcast(notfull);
}

void producer()
{
  char c;
  int i, j;
  printf("In producer\n");
  for (j = 0; j < 500; j++) {
    c = 'a'+cnt;
    /* produce */
    bufput(c);
  }
  printf("producer done\n");
}

void consumer()
{
  char c;
  int i,j;
  for (j = 0; j < 500; j++) {
    bufget(&c);
    /* consume the char */
  }
  printf("consumer done\n");
}

/* Scheduler: alternate priority of c1 and c2 every 0.001 sec */
void sch()
{
  int k;
  thread_t x;
  struct timeval wait;

  x = c1;
  wait.tv_sec = 0;
  wait.tv_usec = 1000;

  for (k = 0; k < 100; k++) {
    lwp_sleep(&wait);
    printf("-----Switching priority\n");
    lwp_setpri(x, MINPRIO);
    if (x.thread_id == c1.thread_id)
      x = c2;
    else
      x = c1;
    lwp_setpri(x, MINPRIO+1);
  }
}
```