UnixWorld Online: Tutorial Article No. 011 Listings

Listing 1. Pseudo-code implementation of restart algorithms.

A. The client functions

```
void newdownload(char* filename)
{
  // The client UI will call this function if user
  // wants to download a file from the beginning,
  // not restart an aborted file transfer procedure
  //
  long tbt = 0;                   // total bytes transferred
  char buf[2048];                 // buffer
  long bt;                        // bytes transferred in
                                  // a single block
  int fd;                         // local file descriptor

  fd = _open(filename, O_WRONLY); // Create a new local file
  bt = svr_read(&buf);        // returns buf size
  while (bt > 0) {

     if (isrestartmarker(buf)){   // if buf is a restart marker
        updatelogservertimestamp(filename,buf); // then update Log file
     } else {
        tbt = tbt + bt;           // increment total bytes transferred
        updatelogbytesize(filename, tbt);
        _write(fd, buf, sizeof(buf));
     };
     bt = svr_read(&buf);
  };
  _close(fd);                     // close local file
}

void restartdownload(char* filename)
{
  // The client UI will call this function if user
  // wants to restart aborted file transfer process
  //
  long tbt = 0;                   // total bytes transferred
  char buf[2048];                 // buffer
  long bt;                        // bytes transferred in a single block
  int fd;                         // local file descriptor
  TIMESTAMP sts;                  // last recorded server timestamp in Log
  TIMESTAMP cts;                  // last recorded client timestamp in Log
  int ret;                        // return code for error recovery

  // Check if the local file size matches that in Transfer Log
  //
  tbt = getlogbytesize(filename);
  sts = getlogservertimestamp(filename);
  cts = getlogclienttimestamp(filename);
  fd = _open(filename, O_APPEND); // Create a new local file
  if (tbt! = _filelength(fd) ||
     cts! = getclientfiletimetstamp(filename)) {
```

```
                                  // local file has changed
                                  // since download
     _close(fd);                  // abort restart procedure
     return();
  };
  ret = svr_restart(filename,     // ask server to start
                    tbt,          // (RPC Call)
                    sts);
  if (ret == -1) goto shutdown;   // error recovery
  bt = svr_read(&buf);
  while (bt > 0) {
                                  // if buf is a restart marker
                                  // then update Log file
     if (isrestartmarker(buf)) {
         updatelogtimestamp(filename, buf);
     } else {
         tbt = tbt + bt;          // increment total bytes
                                  // transferred
         updatelogbytesize(filename, tbt);
         _write(fd, buf, sizeof(buf));
     };
     bt = svr_read(&buf);
  };
  shutdown:
  close(fd);                      // close local file
}

void updatelogservertimestamp(char* filename,char* buf)
{
  int TSLENGTH = 8;               // Timestamp length
  int fd;                         // file descriptor of log file
  LOGSTRUCT ls = NULL;
  char t[8];
  fd = _open(filename, O_RDWR);
  while (read(fd, ls, sizeof(LOGSTRUCT))) {
     if (strcmp(ls.filename, filename) == 0) {
                                  // I found the Log record!
        strncpy(t, ls.filename, TSLENGTH);
                                  // get the timestamp out of buf
        ls.rt = (TIMESTAMP)t;
                                  // cast into timestamp
        _write(fd, ls);
        exit;
     };
  };
  _close(fd);
}

void updatelogclienttimestamp(char* filename, char* buf)
{
  int TSLENGTH = 8;               // Timestamp length
  int fd;                         // file descriptor of log file
  LOGSTRUCT ls = NULL;
  char t[8];
  fd = _open(filename, O_RDWR);
  while (read(fd, ls, sizeof(LOGSTRUCT))) {
     if (strcmp(ls.filename, filename) == 0) {
                                  // I found the Log record!
        strncpy(t, ls.filename, TSLENGTH);
                                  // get the timestamp out of buf
        ls.ct = (TIMESTAMP)t;
```

```
                                  // cast into timestamp
        _write(fd, ls);
        exit;
     };
  };
  _close(fd);
}

updatelogbytesize(char* filename, long tbt)
{
  int fd;                         // file descriptor of log file
  LOGSTRUCT ls = NULL;
  fd = _open(filename, O_RDWR);
  while (read(fd, ls, sizeof(LOGSTRUCT))) {
     if (strcmp(ls.filename, filename) == 0) {
                                  // I found the Log record!
        ls.bytestransferred = tbt;
        _write(fd, ls);
        exit;
     };
  };
  _close(fd);
}

TIMESTAMP getlogservertimestamp(char* filename)
{
  int fd;                         // file descriptor of log file
  LOGSTRUCT ls = NULL;
  fd = _open(filename, O_RDWR);
  while (read(fd, ls, sizeof(LOGSTRUCT))) {
     if (strcmp(ls.filename, filename) == 0) { // I found the Log record!
        return ls.rt;
                                  // return timestamp
     };
  };
  _close(fd);
  return NULL;
}

TIMESTAMP getlogclienttimestamp(char* filename)
{
  int fd;                         // file descriptor of log file
  LOGSTRUCT ls = NULL;
  fd = _open(filename, O_RDWR);
  while (read(fd, ls, sizeof(LOGSTRUCT))) {
     if (strcmp(ls.filename, filename) == 0) {
                                  // I found the Log record!
        return ls.ct;
                                  // return timestamp
     };
  };
  _close(fd);
  return NULL;
}


long getlogbytesize(char *filename)
{
  int fd;                         // file descriptor of log file
  LOGSTRUCT ls = NULL;
  fd = _open(filename, O_RDONLY);
```

```
  while (read(fd, ls, sizeof(LOGSTRUCT))) {
     if (strcmp(ls.filename, filename) == 0) {
                                  // I found the Log record!
        return ls.bytestransferred;
     };
  };
  _close(fd);
  return 0;
}

TIMESTAMP getclientfiletimestamp(char *filename)
{
  TIMESTAMP t = NULL;
  int fd;                         // file descriptor of log file
  char buf[2048];
  fd = _open(filename, O_RDONLY);
  if (fd>0) {
     t = _fstat(fd, &buffer);
     _close(fd);
  };
  return t.st_time;
}
```

B. The server functions

```
static LOGSTRUCT ls;
static int fd;
static int tbt;                   // total bytes transferred
                                  // during this session
static int rc;                    // counter for restart marker intervals
#define RESTART_INTERVAL 100000
int svr_restart(char* filename, long tbt, TIMESTAMP ts)
{
  int c = 2048;
  char buf[2048];
  struct _stat buffer;

  fd = _open(filename, O_RDONLY);
  _fstat(fd, &buffer);
  if (buffer.st_time! = ts) return -1;
                                  // file has changed!
  while (tbt > 0) {               // Otherwise, move file pointer
                                  // by tbt bytes
     if (tbt < c) c = tbt;
     read(fd, buf, c);
     tbt = tbt - c;
  };
}

int svr_read(char* filename,char **buf)
{
  int c = 2048, n;
  struct _stat buffer;

  strcpy(ls.filename, filename);
  if (rc > RESTART_INTERVAL) {
                                  // Send Restart Marker
     rc = 0;                      // every 100K bytes
```

```
      ls.bytestransferred = tbt;
      _fstat(fd, &buffer);
      ls.ts = buffer.st_time;
      buf = (char **)ls;
   } else {                          // else just move the
      n = _read(fd, *buf, c);        // data down the pipe
      tbt = tbt + n;
      rc = rc + n;
   };
   if (n == 0) _close(fd);           // close the file when EOF
   return(strlen(*buf));
};
```

Edited by Becca Thomas / Online Editor / UnixWorld Online / editor@unixworld.com

Go To Content   Search

Last Modified: Sunday, 21-Jan-96 06:09:50 PST