2015/1/7

システムプログラミング演習8

学籍番号：201420694
氏名：星 遼平

1 プログラム

```
#define QSIZE 10

typedef void *(*function_pointer)(void *arg);

typedef struct {
  pthread_mutex_t buf_lock;
  int start;
  int num_full;
  pthread_cond_t notfull;
  pthread_cond_t notempty;
  void *requests[QSIZE];
} circ_buf_t;

typedef struct {
  function_pointer func;
  void *arg;
} request_func;

circ_buf_t cbp = {
  PTHREAD_MUTEX_INITIALIZER,
  0,
  0,
  PTHREAD_COND_INITIALIZER,
  PTHREAD_COND_INITIALIZER,
  NULL
};

typedef struct {
  pthread_t thread;
  int sock;
} connection_info;

char *program_name = "sp6-server";
```

続き

```
static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
static pthread_once_t once = PTHREAD_ONCE_INIT;

int open_accepting_socket(int port) {
  struct sockaddr_in self_addr;
  socklen_t self_addr_size;
  int sock, sockopt;

  memset(&self_addr, 0, sizeof(self_addr));
  self_addr.sin_family = AF_INET;
  self_addr.sin_addr.s_addr = INADDR_ANY;
  self_addr.sin_port = htons(port);
  self_addr_size = sizeof(self_addr);
  sock = socket(PF_INET, SOCK_STREAM, 0);
  if (sock < 0)
    logutil_fatal("accepting socket: %d", errno);
  sockopt = 1;
  if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
        &sockopt, sizeof(sockopt)) == -1)
    logutil_warning("SO_REUSEADDR: %d", errno);
  if (bind(sock, (struct sockaddr *)&self_addr, self_addr_size) < 0)
    logutil_fatal("bind accepting socket: %d", errno);
  if (listen(sock, LISTEN_BACKLOG) < 0)
    logutil_fatal("listen: %d", errno);
  return (sock);
}

void usage(void) {
  fprintf(stderr, "Usage: %s [option]\n", program_name);
  fprintf(stderr, "option:\n");
  fprintf(stderr, "\t-d\t\t\t\t... debug mode\n");
  fprintf(stderr, "\t-p <port>\n");
  exit(1);
}
```

続き

```
void put_cb_data(circ_buf_t *cbp, void *request) {
  pthread_mutex_lock(&cbp->buf_lock);

  while (QSIZE <= cbp->num_full)
    pthread_cond_wait(&cbp->notfull, &cbp->buf_lock);
  cbp->requests[(cbp->start + cbp->num_full) % QSIZE] = request;
  cbp->num_full++;

  pthread_cond_signal(&cbp->notempty);
  pthread_mutex_unlock(&cbp->buf_lock);
}

void *get_cb_data(circ_buf_t *cbp) {
  void *request;

  pthread_mutex_lock(&cbp->buf_lock);
  while (cbp->num_full <= 0)
    pthread_cond_wait(&cbp->notempty, &cbp->buf_lock);
  request = cbp->requests[cbp->start];
  cbp->requests[cbp->start] = NULL;
  cbp->start = (cbp->start + 1) % QSIZE;
  cbp->num_full--;

  pthread_cond_signal(&cbp->notfull);
  pthread_mutex_unlock(&cbp->buf_lock);
  return request;
}

void *enqueue_request(void *data) {
  put_cb_data(&cbp, data);
}

void *connection_cancel(void *conn_info) {
  connection_info *connection = (connection_info *)conn_info;

  sleep(60);

  pthread_cancel(connection->thread);
  close(connection->sock);
  free(connection);
}
```

続き

```
void *dequeue_request(void *data) {
  pthread_t t;
  pthread_t cancel_thread;
  int sock, err;
  connection_info *conn_info;

  request_func *request;
  request = (request_func *)get_cb_data(&cbp);

  sock = *((int *)request->arg);

  if (sock < 0)
    logutil_error("accept error: %d", errno);
  else if ((err = pthread_create(&t, NULL, request->func, request->arg) != 0))
    logutil_error("create_detached_thread: %d", err);

  conn_info = (connection_info *)malloc(sizeof(connection_info));
  conn_info->thread = t;
  conn_info->sock = sock;
  pthread_create(&cancel_thread, NULL, connection_cancel, (void *)conn_info);
}

void *worker(void *data) {
  while (1) {
    dequeue_request(NULL);
  }
}
```

続き

```
void *protocol_main(void *sock_arg) {
  char buf[1024];
  int sock;

  sock = *(int *)sock_arg;

  while (1) {
    read(sock, buf, 1024);

    if (buf[0] == EOF) {
      logutil_info("disconnected\n");
      break;
    }

    write(sock, buf, strlen(buf) + 1);
  }
  close(sock);
}

void main_loop(int accepting_socket) {
  int sock;
  struct sockaddr_in client_addr;
  socklen_t client_addr_size;
  pthread_t t;
  request_func request = { &protocol_main, NULL };

  while (1) {
    client_addr_size = sizeof(client_addr);
    sock = accept(accepting_socket, (struct sockaddr *)&client_addr, &client_addr_size);
    request.arg = (void *)&sock;
    pthread_create(&t, NULL, enqueue_request, (void *)&request);
  }
}
```

続き

```
int main(int argc, char **argv) {
  char *port_number = NULL;
  int ch, sock, server_port = DEFAULT_SERVER_PORT;
  int debug_mode = 0;
  pthread_t worker_1, worker_2, worker_3;

  while ((ch = getopt(argc, argv, "dp:")) != -1) {
    switch (ch) {
      case 'd':
        debug_mode = 1;
        break;
      case 'p':
        port_number = optarg;
        break;
      case '?':
      default:
        usage();
    }
  }
  argc -= optind;
  argv += optind;

  if (port_number != NULL)
    server_port = strtol(port_number, NULL, 0);

  /* server_port で listen し, socket descriptor を sock に代入 */
  sock = open_accepting_socket(server_port);

  if (!debug_mode) {
    logutil_syslog_open(program_name, LOG_PID, LOG_LOCAL0);
    daemon(0, 0);
  }

  pthread_create(&worker_1, NULL, worker, NULL);
  pthread_create(&worker_2, NULL, worker, NULL);
  pthread_create(&worker_3, NULL, worker, NULL);

  main_loop(sock);

  return (0);
}
```

2　実行結果

クライアント

```
$ telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello!
Hello!
Connection closed by foreign host.
$
```

その他のクライアント

```
$ telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
World!
World!
```

3　考察

実行結果で 60 秒たったらスレッドがキャンセルするようにしており，実際に実行結果のクライアントが自動的にコネクションが切れていることがわかる．その他 4 つのクライアントに接続しているが，他のものは継続してコネクションし続けており，60 秒経過したコネクションごとに接続が切れている．

実際にスレッドプールを利用することで資源に制限がある場合に有効活用できることがわかった．今回サーバとクライアントとの通信によるプログラムを演習で作成したが，実際の HTTP サーバなどのを作成したときは，ユーザのコネクションリクエストをずっと待たせるわけにはいかず，交互に接続するなどの工夫が必要になるだろうと考えた．また，通常の Web サービスなどのアカウント登録時にメール認証する場合があるが，その場合メール送信処理は非同期処理で行われることがあり，その場合はスレッドプールが利用できるだろうと思った．

4　授業の感想

POSIX でスレッドをキャンセルするときなどに様々手法があり，その上で C 言語では `pthread_cancel` というライブラリが用意されているというようにメリットとデメリットを示されていて理解しやすかった．個人的に難易度が上がってきたと感じており，演習を通して復讐していきたいと感じた．