Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Многопоточная реализация RPC для решения задачи «производители-потребители»

Москва, 2024

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Особенностью реализованного сервера является то, что выполнение нескольких клиентов на одной машине приводят к ошибкам, так как ответы сервера клиентам перемешиваются. В случае же запуска клиентов на разных ip-адресах разработанное ПО работает корректно.

Листинг 1 — «Файл pc.x для генерации сервера и скелетонов»

```
/*
* filename: pc.x
* function: Define constants, non-standard data types and the calling
    process in remote calls
*/


program PC_PROG
{
  version PC_VERSION
  {
    char consume(void) = 1;
    char produce(void) = 2;
  } = 1; /* Version number = 1 */
} = 0x20000001; /* RPC program number */
```

Для генерации файлов используется команда:

```
rpcgen pc.x -a -M
```

Листинг 2 — «Модифицированный файл pc_svc.c с кодом многопоточного rpc сервер»

```
/*
* Please do not edit this file.
* It was generated using rpcgen.
*/

#include "pc.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
```

```
#define SIG_PF void(*)(int)
#endif

pthread_t p_thread;
pthread_attr_t attr;

// У всех функций нет аргументов, создан для rpc
union argument_t {
  int fill;
};
void*
serv_request(void *data)
{
  struct thr_data
  {
    struct svc_req *rqstp;
    SVCXPRT *transp;
    union argument_t argument;
  } *ptr_data;
  union argument_t argument;
  union {
    char consume_1_res;
    char produce_1_res;
  } result;
  bool_t retval;
  xdrproc_t _xdr_argument, _xdr_result;
  bool_t (*local)(char *, void *, struct svc_req *);
  ptr_data = (struct thr_data *) data;
  struct svc_req *rqstp = ptr_data->rqstp;
  register SVCXPRT *transp = ptr_data->transp;
  argument = ptr_data->argument;
  switch (rqstp->rq_proc) {
    case NULLPROC:
    (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
    return NULL;
    case consume:
    _xdr_argument = (xdrproc_t) xdr_void;
    _xdr_result = (xdrproc_t) xdr_char;
    local = (bool_t (*) (char *, void *,  struct svc_req *))
       consume_1_svc;
    break;
```

```
    case produce:
    _xdr_argument = (xdrproc_t) xdr_void;
    _xdr_result = (xdrproc_t) xdr_char;
    local = (bool_t (*) (char *, void *,  struct svc_req *))
       produce_1_svc;
    break;
    default:
    svcerr_noproc (transp);
    return NULL;
  }
  retval = (bool_t) (*local)((char *)&argument, (void *)&result, rqstp);
  if (retval > 0 && !svc_sendreply(transp, (xdrproc_t) _xdr_result, (
     char *)&result)) {
    svcerr_systemerr (transp);
  }
  if (!svc_freeargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &
     argument)) {
    fprintf (stderr, "%s", "unable to free arguments");
    exit (1);
  }
  if (!pc_prog_1_freeresult (transp, _xdr_result, (caddr_t) &result))
  fprintf (stderr, "%s", "unable to free results");

  return NULL;
}

static void
pc_prog_1(struct svc_req *rqstp, register SVCXPRT *transp) {
  xdrproc_t _xdr_argument;

  // выделение структуры, для передачи обработчику в отдельном потоке
  struct data_str
  {
    struct svc_req *rqstp;
    SVCXPRT *transp;
    union argument_t argument;
  } *data_ptr=(struct data_str*)malloc(sizeof(struct data_str));
  if (data_ptr == NULL) {
    fprintf (stderr, "%s", "unable to allocate memory.");
    exit(1);
  }
```

```
  // получение аргументов вызова
  switch (rqstp->rq_proc) {
    case NULLPROC:
    (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
    return ;
    break;
    case produce:
    _xdr_argument = (xdrproc_t) xdr_void;
    break;
    case consume:
    _xdr_argument = (xdrproc_t) xdr_void;
    break;
    default:
    svcerr_noproc (transp);
    return ;
    break;
  }
  // svc_getargs не потокобезопасна, поэтому, если аргументы есть, их по
     лучение происходит на этом этапе
  memset ((char *)&data_ptr->argument, 0, sizeof (data_ptr->argument));
  if (!svc_getargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &
     data_ptr->argument)) {
    svcerr_decode (transp);
    return;
  }
  data_ptr->rqstp = rqstp;
  data_ptr->transp = transp;

  // Создание detached потока с обработчиком
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
  pthread_create(&p_thread,&attr,serv_request,(void *)data_ptr);
}

int
main (int argc, char **argv)
{
  register SVCXPRT *transp;

  // Инициализация семафоров и буфера
  if (init_pc() != 0) {
```

```
    fprintf (stderr, "%s", "unable to initialize.");
    exit(1);
  }
  pmap_unset (PC_PROG, PC_VERSION);
  transp = svcudp_create(RPC_ANYSOCK);
  if (transp == NULL) {
    fprintf (stderr, "%s", "cannot create udp service.");
    exit(1);
  }
  if (!svc_register(transp, PC_PROG, PC_VERSION, pc_prog_1, IPPROTO_UDP)
     ) {
    fprintf (stderr, "%s", "unable to register (PC_PROG, PC_VERSION, udp
       ).");
    exit(1);
  }
  transp = svctcp_create(RPC_ANYSOCK, 0, 0);
  if (transp == NULL) {
    fprintf (stderr, "%s", "cannot create tcp service.");
    exit(1);
  }
  if (!svc_register(transp, PC_PROG, PC_VERSION, pc_prog_1, IPPROTO_TCP)
     ) {
    fprintf (stderr, "%s", "unable to register (PC_PROG, PC_VERSION, tcp
       ).");
    exit(1);
  }
  svc_run ();
  fprintf (stderr, "%s", "svc_run returned");
  exit (1);
  /* NOTREACHED */
}
```

Листинг 3 — «Файл скелетона pc_server.c с реализацией функций производителей и потребителей»

```
/*
* This is sample code generated by rpcgen.
* These are only templates and you can use them
* as a guideline for developing your own functions.
*/

#include "pc.h"
```

```
#include <sys/sem.h>

#define SEMAPHORE_EMPTY 0
#define SEMAPHORE_FULL 1
#define SEMAPHORE_BINARY 2

#define p -1
#define v 1

struct sembuf start_consume[] = {{SEMAPHORE_FULL, p, 0}, {
   SEMAPHORE_BINARY, p, 0}};
struct sembuf end_consume[] = {{SEMAPHORE_BINARY, v, 0}, {
   SEMAPHORE_EMPTY, v, 0}};

struct sembuf start_produce[] = {{SEMAPHORE_EMPTY, p, 0}, {
   SEMAPHORE_BINARY, p, 0}};
struct sembuf end_produce[] = {{SEMAPHORE_BINARY, v, 0}, {SEMAPHORE_FULL
   , v, 0}};

#define BUFFER_SIZE 1000
char *buf;
char *current_consume;
char *current_produce;
char next_letter;
int semid;

int init_pc() {
  buf = malloc(sizeof(char) * BUFFER_SIZE);
  if (buf == NULL) {
    perror("malloc");
    return 1;
  }
  current_consume = buf;
  current_produce = buf;
  next_letter = 'a';
  semid = semget(IPC_PRIVATE, 3, IPC_CREAT | 0666);
  if (semid == -1) {
    perror("semget");
    return 1;
  }
  if (semctl(semid, SEMAPHORE_EMPTY, SETVAL, BUFFER_SIZE) == -1) {
```

```
    perror("semctl");
    return 1;
  }
  if (semctl(semid, SEMAPHORE_BINARY, SETVAL, 1) == -1) {
    perror("semctl");
    return 1;
  }
  if (semctl(semid, SEMAPHORE_FULL, SETVAL, 0) == -1) {
    perror("semctl");
    return 1;
  }
  return 0;
}

bool_t
consume_1_svc(void *argp, char *result, struct svc_req *rqstp)
{
  bool_t retval;
  int err = semop(semid, start_consume, 2);
  if (err == -1) {
    perror("semop\n");
    exit(1);
  }
  *result = *current_consume;
  current_consume++;
  if (current_consume - buf == BUFFER_SIZE) {
    current_consume = buf;
  }
  err = semop(semid, end_consume, 2);
  if (err == -1) {
    perror("semop\n");
    exit(1);
  }
  return TRUE;
}

bool_t
produce_1_svc(void *argp, char *result, struct svc_req *rqstp)
{
  bool_t retval;
  int err = semop(semid, start_produce, 2);
```

```
  if (err == -1) {
    perror("semop\n");
    exit(1);
  }
  *current_produce = next_letter;
  *result = next_letter;
  next_letter++;
  if (next_letter > 'z') {
    next_letter = 'a';
  }
  current_produce++;
  if (current_produce - buf == BUFFER_SIZE) {
    current_produce = buf;
  }
  err = semop(semid, end_produce, 2);
  if (err == -1) {
    perror("semop\n");
    exit(1);
  }
  return TRUE;
}

int
pc_prog_1_freeresult (SVCXPRT *transp, xdrproc_t xdr_result, caddr_t
    result)
{
  xdr_free (xdr_result, result);
  /*
  * Insert additional freeing code here, if needed
  */
  return 1;
}
```

Листинг 4 — «Файл скелетона pc_client.c с клиентам потребителями и производителями»

```
/*
* This is sample code generated by rpcgen.
* These are only templates and you can use them
* as a guideline for developing your own functions.
*/

#include "pc.h"
#include <string.h>
```

```c
#include <unistd.h>
#include <stdlib.h>

void
pc_prog_consume(char *host)
{
  CLIENT *clnt;
  enum clnt_stat retval_1;
  char result_1;
  char *produce_1_arg;
  #ifndef DEBUG
  clnt = clnt_create (host, PC_PROG, PC_VERSION, "tcp");
  if (clnt == NULL) {
    clnt_pcreateerror (host);
    exit (1);
  }
  #endif  /* DEBUG */
  srand(getpid());
  while (1) {
    sleep(rand() % 3 + 1);
    retval_1 = consume_1((void*)&produce_1_arg, &result_1, clnt);
    if (retval_1 != RPC_SUCCESS) {
      clnt_perror (clnt, "call failed");
    }
    printf("get %c\n", result_1);
  }
  #ifndef DEBUG
  clnt_destroy (clnt);
  #endif   /* DEBUG */
}

void
pc_prog_produce(char *host)
{
  CLIENT *clnt;
  enum clnt_stat retval_1;
  char result_1;
  char *produce_1_arg;
  #ifndef DEBUG
  clnt = clnt_create (host, PC_PROG, PC_VERSION, "tcp");
  if (clnt == NULL) {
```

```
    clnt_pcreateerror (host);
    exit (1);
  }
  #endif  /* DEBUG */
  srand(getpid());
  while (1) {
    sleep(rand() % 3 + 1);

    retval_1 = produce_1((void*)&produce_1_arg, &result_1, clnt);
    if (retval_1 != RPC_SUCCESS) {
      clnt_perror (clnt, "call failed");
    }
    printf("put %c\n", result_1);
  }
  #ifndef DEBUG
  clnt_destroy (clnt);
  #endif   /* DEBUG */
}


int
main (int argc, char *argv[])
{
  char *host;

  if (argc < 3) {
    printf ("usage: %s server_host pc_type\n", argv[0]);
    printf ("pc_type = 1 - consumer\n");
    printf ("pc_type = 2 - producer\n");
    exit (1);
  }
  if (strcmp(argv[2], "1") != 0 && strcmp(argv[2], "2") != 0) {
    printf ("usage: %s server_host pc_type\n", argv[0]);
    printf ("pc_type = 1 - consumer\n");
    printf ("pc_type = 2 - producer\n");
    exit (1);
  }
  host = argv[1];
  if (strcmp(argv[2], "1") == 0) {
    pc_prog_consume (host);
  }
```

```
    else {
      pc_prog_produce (host);
    }
    exit (0);
  }
```

Файл «pc_clnt.c» с кодом клиента не изменяется, а в файл «pc.h» необходимо добавить строчку:

```
int init_pc();
```

В «Makefile.pc» нужно изменить две переменные:

```
CFLAGS = -g -I/usr/include/tirpc
LDLIBS = -lnsl -lpthread -ltirpc
```