|  |
| --- |
| **Thread1.c**  #include <stdio.h>  #include <pthread.h>  #include <unistd.h>  void \* work(void \* ptr)  {  int i;  for (i = 0; i < 10; i++)  {  printf("%d", (int)ptr);  usleep(1000);  }  pthread\_exit(0);  }  int main(int argc, char \*\* argv)  {  pthread\_t t0, t1;  pthread\_create(&t0, 0, work, (void \*)0);  pthread\_create(&t1, 0, work, (void \*)1);  pthread\_join(t0, 0);  pthread\_join(t1, 0);  return 0;  } |
| **Thread2.c**  #include <stdio.h>  #include <pthread.h>  #include <unistd.h>    pthread\_mutex\_t mtx;  void print(int thread, int i)  {  pthread\_mutex\_lock(&mtx);  printf("thread %d: %d\n", thread, i);  pthread\_mutex\_unlock(&mtx);  }  void \* work(void \* ptr)  {  int i;  for (i = 0; i < 10; i++)  {  print((int)ptr, i);  usleep(1000);  }  pthread\_exit(0);  }  int main(int argc, char \*\* argv)  {  pthread\_t t0, t1;  pthread\_mutex\_init(&mtx, 0);  pthread\_create(&t0, 0, work, (void \*)0);  pthread\_create(&t1, 0, work, (void \*)1);  pthread\_join(t0, 0);  pthread\_join(t1, 0);  pthread\_mutex\_destroy(&mtx);  return 0;  } |
| **Thread3.c**  #include <stdio.h>  #include <pthread.h>  pthread\_mutex\_t mtx;  pthread\_cond\_t cond;  int how\_many = 10;  int pool = 0;  void \* producer(void \* ptr)  {  while (how\_many > 0)  {  pthread\_mutex\_lock(&mtx);  printf("producer: %d\n", how\_many);  pool = how\_many;  how\_many--;  pthread\_mutex\_unlock(&mtx);  pthread\_cond\_signal(&cond);  }  pthread\_exit(0);  }  void \* consumer(void \* ptr)  {  while (how\_many > 0)  {  pthread\_mutex\_lock(&mtx);  pthread\_cond\_wait(&cond, &mtx);  printf("consumer: %d\n", pool);  pool = 0;  pthread\_mutex\_unlock(&mtx);  }  pthread\_exit(0);  }  int main(int argc, char \*\* argv)  {  pthread\_t prod, cons;  pthread\_mutex\_init(&mtx, 0);  pthread\_cond\_init(&cond, 0);  pthread\_create(&cons, 0, consumer, 0);  pthread\_create(&prod, 0, producer, 0);  pthread\_join(prod, 0);  pthread\_join(cons, 0);  pthread\_cond\_destroy(&cond);  pthread\_mutex\_destroy(&mtx);  return 0;  } |
| **Client.c**  #include <stdio.h>  #include <sys/socket.h>  #include <netdb.h>  #include <unistd.h>  #include <string.h>  int main(int argc, char \*\* argv)  {  int port;  int sock = -1;  struct sockaddr\_in address;  struct hostent \* host;  int len;    /\* checking commandline parameter \*/  if (argc != 4)  {  printf("usage: %s hostname port text\n", argv[0]);  return -1;  }  /\* obtain port number \*/  if (sscanf(argv[2], "%d", &port) <= 0)  {  fprintf(stderr, "%s: error: wrong parameter: port\n", argv[0]);  return -2;  }  /\* create socket \*/  sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (sock <= 0)  {  fprintf(stderr, "%s: error: cannot create socket\n", argv[0]);  return -3;  }  /\* connect to server \*/  address.sin\_family = AF\_INET;  address.sin\_port = htons(port);  host = gethostbyname(argv[1]);  if (!host)  {  fprintf(stderr, "%s: error: unknown host %s\n", argv[0], argv[1]);  return -4;  }  memcpy(&address.sin\_addr, host->h\_addr\_list[0], host->h\_length);  if (connect(sock, (struct sockaddr \*)&address, sizeof(address)))  {  fprintf(stderr, "%s: error: cannot connect to host %s\n", argv[0], argv[1]);  return -5;  }  /\* send text to server \*/  len = strlen(argv[3]);  write(sock, &len, sizeof(int));  write(sock, argv[3], len);  /\* close socket \*/  close(sock);  return 0;  } |
| **Server.c**  #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <sys/socket.h>  #include <linux/in.h>  #include <unistd.h>  typedef struct  {  int sock;  struct sockaddr address;  int addr\_len;  } connection\_t;  void \* process(void \* ptr)  {  char \* buffer;  int len;  connection\_t \* conn;  long addr = 0;  if (!ptr) pthread\_exit(0);  conn = (connection\_t \*)ptr;  /\* read length of message \*/  read(conn->sock, &len, sizeof(int));  if (len > 0)  {  addr = (long)((struct sockaddr\_in \*)&conn->address)->sin\_addr.s\_addr;  buffer = (char \*)malloc((len+1)\*sizeof(char));  buffer[len] = 0;  /\* read message \*/  read(conn->sock, buffer, len);  /\* print message \*/  printf("%d.%d.%d.%d: %s\n",  (int)((addr ) & 0xff),  (int)((addr >> 8) & 0xff),  (int)((addr >> 16) & 0xff),  (int)((addr >> 24) & 0xff),  buffer);  free(buffer);  }  /\* close socket and clean up \*/  close(conn->sock);  free(conn);  pthread\_exit(0);  }  int main(int argc, char \*\* argv)  {  int sock = -1;  struct sockaddr\_in address;  int port;  connection\_t \* connection;  pthread\_t thread;  /\* check for command line arguments \*/  if (argc != 2)  {  fprintf(stderr, "usage: %s port\n", argv[0]);  return -1;  }  /\* obtain port number \*/  if (sscanf(argv[1], "%d", &port) <= 0)  {  fprintf(stderr, "%s: error: wrong parameter: port\n", argv[0]);  return -2;  }  /\* create socket \*/  sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (sock <= 0)  {  fprintf(stderr, "%s: error: cannot create socket\n", argv[0]);  return -3;  }  /\* bind socket to port \*/  address.sin\_family = AF\_INET;  address.sin\_addr.s\_addr = INADDR\_ANY;  address.sin\_port = htons(port);  if (bind(sock, (struct sockaddr \*)&address, sizeof(struct sockaddr\_in)) < 0)  {  fprintf(stderr, "%s: error: cannot bind socket to port %d\n", argv[0], port);  return -4;  }  /\* listen on port \*/  if (listen(sock, 5) < 0)  {  fprintf(stderr, "%s: error: cannot listen on port\n", argv[0]);  return -5;  }  printf("%s: ready and listening\n", argv[0]);    while (1)  {  /\* accept incoming connections \*/  connection = (connection\_t \*)malloc(sizeof(connection\_t));  connection->sock = accept(sock, &connection->address, &connection->addr\_len);  if (connection->sock <= 0)  {  free(connection);  }  else  {  /\* start a new thread but do not wait for it \*/  pthread\_create(&thread, 0, process, (void \*)connection);  pthread\_detach(thread);  }  }    return 0;  } |