Question: C Program void* handleClient(void* vPtr): (Or

C Program

void* handleClient(void* vPtr):

(Or whatever you call your function that runs a thread for the client.)

1. The thread id and the file descriptor are passed, but they come in as a void* pointer.
   - Use another pointer to cast back to int*
   - Save the file descriptor and thread number in local vars
   - free() the memory
2. Get an integer in *network endianness* from the client, and convert it to its own endianness. Call this the filenameLen.
3. Get exactly filenameLen chars from the client. Put this in filenameBuffer. Add an ending '\0' at the end of what you have read().
4. Call didFindFile() in an if statement.
5. If didFindFile() returns true then:
   1. Sends the integer 1 in *network endianness* to the client to say *"I found the file"*
   2. Sends the length of the pathname (filePathLen, which is strlen(filePathBuffer)) to the client in *network endianness*.
   3. Sends exactly that number of chars from filePathBuffer.
   4. Opens that file and read()s up to BUFFER_LEN chars.
   5. Sends exactly those chars that were read().
   6. close() the file descriptor for read()ing the file.
6. If didFindFile() returns false then:
   1. Sends the integer 0 in *network endianness* to the client to say *"I did not find the file"*
7. close() the file descriptor for talking to the client.

```
//  PURPOSE:  To handle the client being communicated with socket file
//      descriptor '*(int*)vPtr'.  Returns 'NULL'.
void*         handleClient    (void*          vPtr
void            handleClient     (void            vPtr
                                 )
{
  //  I.  Application validity check:

  //  II.  Handle client:
  //  II.A.  Get file descriptor:
  int*     intArray;
  int      clientFd;
  int      threadNum;

  //  YOUR CODE HERE:
  printf("Thread %d starting.\n",threadNum);

  //  II.B.  Read filename:
  char  filenameBuffer[BUFFER_LEN];
  char  filePathBuffer[BUFFER_LEN];
  int   filenameLen;
  int   temp;
  int   readLen;

  //  YOUR CODE HERE:

  if  ( didFindFile(filePathBuffer,BUFFER_LEN,".",filenameBuffer) )
  {
    int         fileFd;
    char        fileBuffer[BUFFER_LEN];
    int         filePathLen     = strlen(filePathBuffer);

    //  YOUR CODE HERE:
  }
  else
  {
    //  YOUR CODE HERE:
  }

  //  III.  Finished:
  printf("Thread %d quitting.\n",threadNum);
  //  YOUR CODE HERE:
  return(NULL);
}


//  PURPOSE:  To run the server by 'accept()'-ing client requests from
//      'listenFd' and doing them.
void        doServer    (int        listenFd
                )
{
  //  I.  Application validity check:

  //  II.  Server clients:
  pthread_t        threadId;
  pthread_attr_t    threadAttr;
  int            threadCount    = 0;
  //  YOUR CODE HERE:

  while  (1)
  {
    int* clientFdPtr     = (int*)malloc(2*sizeof(int));
    //  YOUR CODE HERE:

    int connfd = accept(listenFd, NULL, Null);
```

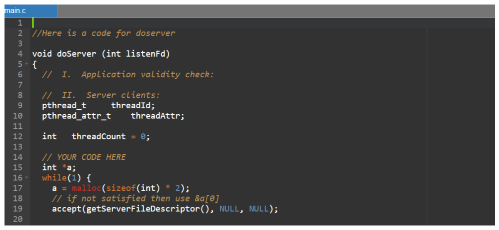
```
        clientFdPtr[0] = listenFd;
        clientFdPtr[1] = threadCount++;

        pthread_attr_init(&threadAttr);
        pthread_attr_setdetachstate(&threadAttr, PTHREAD_CREATE_DETACHED);

        pthread_create(&threadId, &threadAttr, handleClient, (void *)clientFdPtr);

        pthread_join(threadId, NULL);
  }

  pthread_attr_destroy(&threadAttr);

  //  III.  Finished:
```

2 Comments

Expert Answer

Anonymous
answered this

```
//Here is a code for doserver

void doServer (int listenFd)
{
  //  I.  Application validity check:

  //  II.  Server clients:
  pthread_t      threadId;
  pthread_attr_t    threadAttr;

  int   threadCount = 0;

  // YOUR CODE HERE
  int *a;
  while(1) {
    a = malloc(sizeof(int) * 2);
    // if not satisfied then use &a[0]
    accept(getServerFileDescriptor(), NULL, NULL);

    // 2.
    a[0] = getServerFileDescriptor();

    // 3.
    a[1] = threadCount++;

    // ALL 4
    pthread_attr_init(&threadAttr);
    pthread_attr_setdetachstate(&threadAttr, PTHREAD_CREATE_DETACHED);
    pthread_create(&threadId, &threadAttr, handleClient, &a[0]);

    pthread_join(threadId, NULL);
    pthread_attr_destroy(&threadAttr);

   }

}

//Here's my handle Client method:

void* handleClient(void* vPtr) {

  // Use another pointer to cast back to int*
  // Save the file descriptor and thread number in local vars
  // free() the memory

  // code.

printf("&a=%p\n", (void *) &a);

printf("castMe=%p\n", (void *) castMe);

int * const numbers = vPtr;

  free(vPtr);

  //  II.B.  Read command:
  char  buffer[BUFFER_LEN];
  char  command;
  int fileNum;

  int fd = castMe[0];
  int threadNum = castMe[1];

  char  text[BUFFER_LEN];
  int   shouldContinue  = 1;

  while  (shouldContinue)
  {
    text[0] = '\0';

    read(fd,buffer,BUFFER_LEN);
    printf("Thread %d received: %s\n",threadNum,buffer);
    sscanf(buffer,"%c %d \"%[^\"]\"",&command,&fileNum,text);

    //printf("Thread %d quitting.\n",threadNum);
    return(NULL);

    // YOUR CODE HERE

  }
```



main.c

```

//Here is a code for doserver

void doServer (int listenFd)
{
  //  I.  Application validity check:

  //  II.  Server clients:
  pthread_t     threadId;
  pthread_attr_t    threadAttr;

  int   threadCount = 0;

  // YOUR CODE HERE
  int *a;
  while(1) {
    a = malloc(sizeof(int) * 2);
    // if not satisfied then use &a[0]
    accept(getServerFileDescriptor(), NULL, NULL);

```

0 Comments

Questions viewed by other students

Was this answer helpful? 0 2

Implementing doServer(int listenFd): doServer() should have a loop in which it waits for a client to connect to listenFd. When a client does, it should: malloc() enough memory for 2 integers put the file descriptor from accept() in one of those spaces put the value of threadCount in the other space, and increment threadCount Make a detached thread to handle this new client. I...

See answer

please help me finish the function void playTennis (int toClientFd) --- see below OVERVIEW: Write a server that is able to: Bind a port and serve as a server Wait for clients to connect to the socket and make client-handling threads Receive input from a socket fork() a child to execute referee when told to send back the tennis score in network endian. wait() for the child process...

See answer

Show more