

|  |  |
| --- | --- |
| IPO表 | |
| 系统：CNS服务器 | 作者：Ljf |
| 模块：主线程（main） | 日期：2018/07/11 |
| 编号： |  |
| 被调用： | 调用：  create\_service\_listener()  sign\_init()  cond\_mutexInit()  thread\_init()  ND\_SVSemPost();  ND\_SVTaskPost(); |
| 输入：  获得所有活动文件描述符 | 输出：   1. 增加接收线程信号量，同时唤醒接收线程 2. 把任务加到任务队列中，同时唤醒工作线程 |
| 处理：  create\_service\_listener(addr,port);  sign\_init();  cond\_mutex\_init();  thread\_init();  While(true)  {  struct epoll\_event ev[ev\_length];  int n = epoll\_wait(epollfd, ev, ev\_length, TIME\_OUT);  for (int i = 0; i < m; ++i)  {  if (ev[i].data.fd == listenfd)  {  Pthread\_mutex\_lock(&accept\_mutex);  ND\_SVSemPost();  pthread\_cond\_signal(&accept\_cond);  Pthread\_mutex\_unlock(&accept\_mutex);  }  else if(ev[i].events)  {  Pthread\_mutex\_lock(&worker\_mutex);  ND\_SVTaskPost(ev[i].data.fd);  pthread\_cond\_signal(&worker\_cond);  Pthread\_mutex\_unlock(&worker\_mutex);  }  } | |
| 局部数据元素： | 注释： |

|  |  |
| --- | --- |
| IPO表 | |
| 系统：CNS服务器 | 作者：Ljf |
| 模块：接收新连接线程（accept\_thread） | 日期：2018/07/11 |
| 编号： |  |
| 被调用： | 调用：  ND\_SVSemPend()  ND\_SVFdInit() |
| 输入：  新的客户连接描述符 | 输出：  1.初始化新客户用户结构 |
| 处理：  While(true)  {  Pthread\_mutex\_lock(&accept\_mutex);  While((!ND\_SVSemPend)  Phread\_cond\_wait(&accept\_cond,&accept\_mutex);  struct sockaddr\_in clientaddr;  socklen\_t addrlen;  int newfd = ::accept(listenfd, (struct sockaddr \*)&clientaddr, &addrlen);  Pthread\_mutex\_unlock(&accept\_mutex);  ND\_SVFdInit(user[newfd],newfd);  } | |
| 局部数据元素：  Clientaddr 新连接客户端地址  Addrlen 新连接客户端地址长度  Newdf 新连接客户端描述符  E 给新描述符添加的epoll事件 | 注释： |

|  |  |
| --- | --- |
| IPO表 | |
| 系统：CNS服务器 | 作者：Ljf |
| 模块：工作线程（worker\_thread） | 日期：2018/07/11 |
| 编号： |  |
| 被调用： | 调用：  ND\_SVTaskPend  RecvFromClient  WriteToClient |
| 输入：   1. 从任务队列中获取新的任务 2. 读取客户的新数据 | 输出：  1.把数据写给用户 |
| 处理：  While(true)  {  Pthread\_mutex\_lock(&worker\_mutex);  While((user= ND\_SVTaskPend)==NULL)  Phread\_cond\_wait(&worker\_cond,&worker\_mutex);  Pthread\_mutex\_unlock(&worker\_mutex);  RecvFromClient();  WriteToClient();  } | |
| 局部数据元素：  User 返回用户数据结构 | 注释： |

|  |  |
| --- | --- |
| 全局变量： | |
| Accept\_cond | 接收线程的条件变量 |
| Accept\_mutex | 接收线程的信号量 |
| Worker\_cond | 工作线程的条件变量 |
| Worker\_mutex | 工作线程的信号量 |
| listen\_fd | 监听文件描述符 |
| epoll\_fd | epoll事件描述符 |
| ListenSemCtr | 监听可收取新连接的个数 |
| User[Max\_USER\_NUM] | 用户数组 |

|  |  |
| --- | --- |
| 宏定义 | |
| WORKER\_THREAD\_NUM | 工作线程数量 |
| ACCEPT\_THREAD\_NUM | 接收线程数量 |
| EPOLL\_EVENT\_NUM | Epoll事件数量 |
| MAX\_USER\_NUM | 最大连接用户数量 |
| MAX\_IOVEC\_NUM | 最大iovec结构体数量 |
| BUF\_SIZE | 用户接收发送缓冲区大小 |
| DBCONNECT\_NUM | 数据库连接数量 |
| SLOT\_NUM | 时间轮槽数量 |
| SLOT\_INTERVAL | 时间轮槽时间间隔 |

|  |  |  |
| --- | --- | --- |
| 类型 | ND\_SV\_USER | 用户数据结构 |
| Int32 | Fd | 用户连接的文件描述符 |
| Int8 | \*Readbuf | 读缓存区 |
| struct iovec | \*Readiovec | 指向读iovec结构的指针 |
| Int8 | NumReadiov | 读取iovec的个数 |
| Int8 | \*Writebuf | 写缓存区 |
| struct iovec | \*Writeiovec | 指向写iovec结构的指针 |
| Int8 | NumWriteiov | 写iovec的个数 |
| nd\_sv\_user | \*NextPtr | 指向下一个用户数据结构 |
| nd\_sv\_timer | \*timer | 指向所拥有的定时器 |

|  |  |  |
| --- | --- | --- |
| 类型 | ND\_SV\_Q | 任务消息队列数据结构 |
| nd\_sv\_user | \*InPtr | 指向最后用户数据结构 |
| nd\_sv\_user | \*OutPtr | 指向首个用户数据结构 |
| Int32 | NumEntries | 已存在的用户个数 |

|  |  |  |
| --- | --- | --- |
| 数据类型 | ND\_SV\_DBC\_POOL | 数据库空闲连接池 |
| ND\_SV\_DBC | \*NextPtr | 池中指向下一个空闲的数据库连接 |
| Int8 | NbrFree | 池中空闲的数据库连接数量 |
| Int8 | NbrUsed | 池中已使用的数据库连接数量 |

|  |  |  |
| --- | --- | --- |
| 数据类型 | ND\_SV\_DBC | 数据库连接数据结构 |
| MYSQL | \*DBC | 指向当前数据库连接对象 |
| ND\_SV\_DBC | \*NextPtr | 指向下一个数据库连接数据结构 |

|  |  |  |
| --- | --- | --- |
| 数据类型 | ND\_SV\_TIMER | 定时器数据结构 |
| Int32 | Rotation | 定时器剩余圈数 |
| Int32 | Time\_Slot | 定时器所在时间轮的槽数 |
| void(\*)(nd\_sv\_user\*) | Cb\_Func | 定时器结束回调函数 |
| nd\_sv\_user | \*User\_Data | 指向定时器所属用户信息 |
| nd\_sv\_timer | \*NextPtr | 指向下一个定时器 |
| nd\_sv\_timer | \*PrevPtr | 指向上一个定时器 |

|  |  |  |
| --- | --- | --- |
| 数据类型 | ND\_SV\_WHEEL | 时间轮数据结构 |
| nd\_sv\_timer\* | SlotArrayPtr | 指向时间轮上每个槽所在的定时器链表 |
| Int32 | Cur\_Slot | 当前所在的槽 |