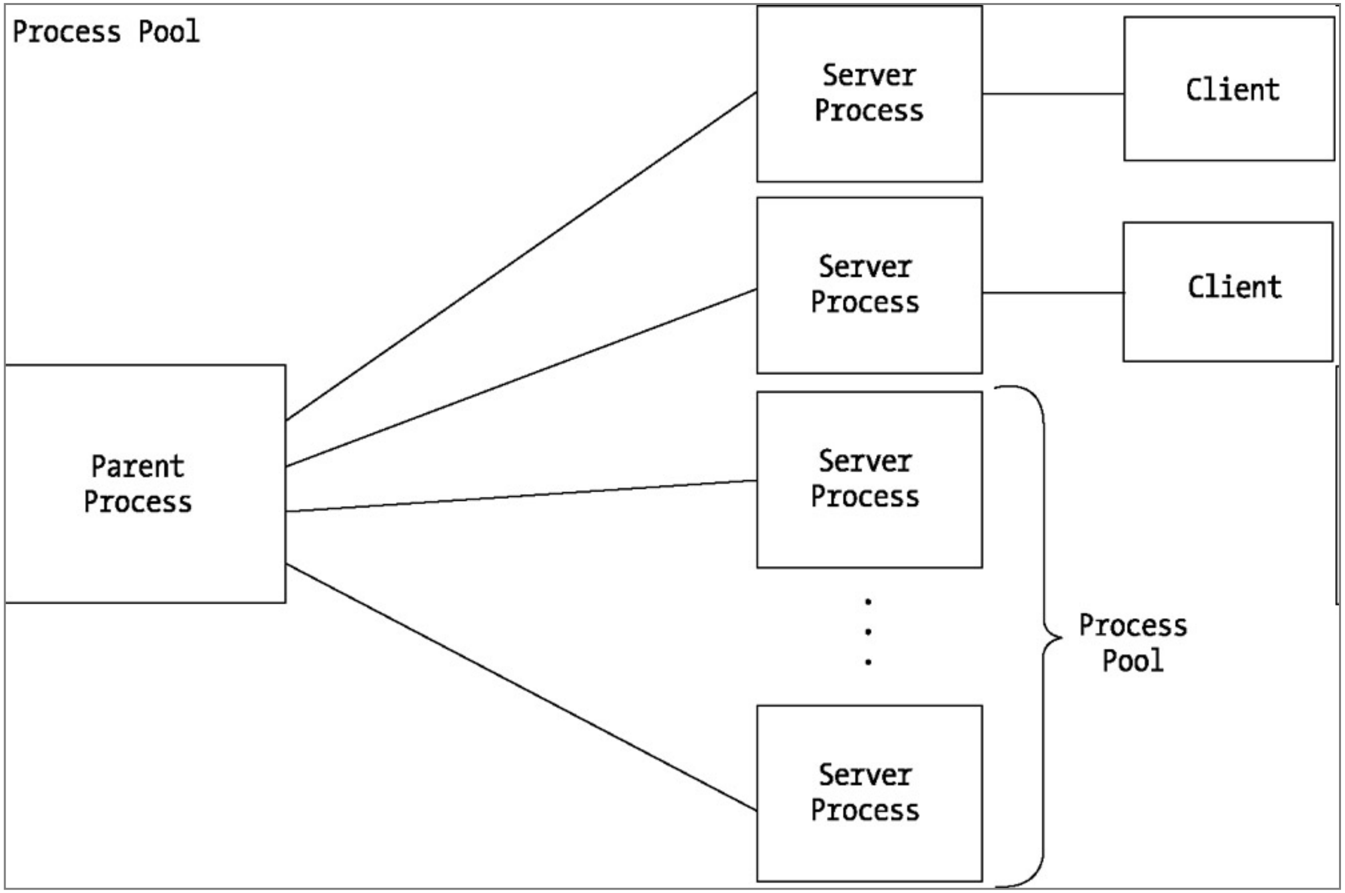
Preforked and prethreaded servers

- Traditional concurrent server model:
  - Fork a child after accepting a new client connection.
  - Good enough for low traffic services.
- For very high-load servers
  - web servers handling thousands of requests per minute
  - the cost of creating a new child (or even thread) for each client imposes a significant burden on the server.
- Instead of creating a new child process (or thread) for each client, the server precreates a fixed number of child processes (or threads) on startup.
  - Each child (thread) handles a new client. After completing one client, it accepts another connection.

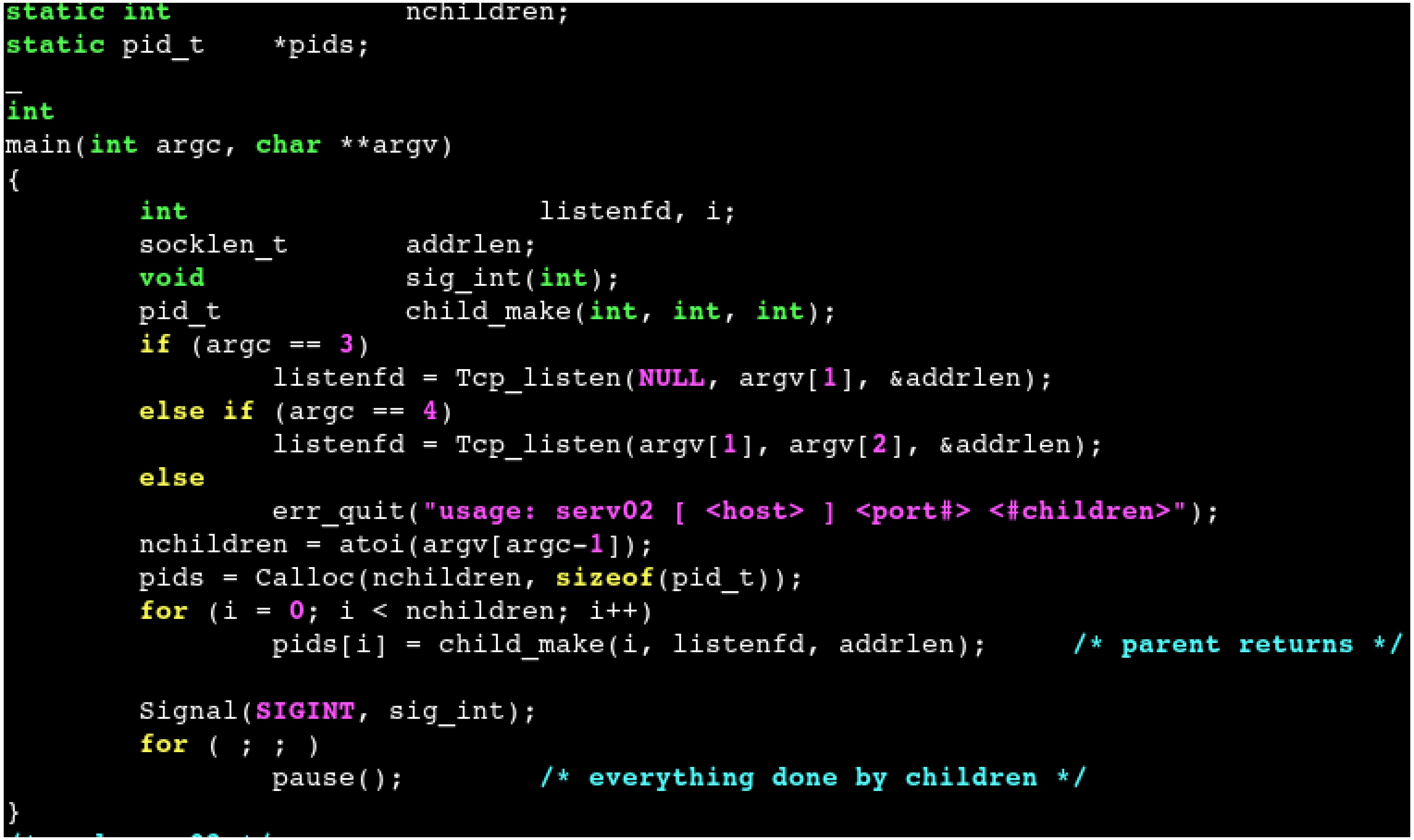
Preforking

- Different models
  - Child calls accept()
    - TCP Preforked Server, No Locking Around accept
    - TCP Preforked Server, Thread Locking Around accept
  - Parent calls accept() and passes the descriptor to child
    - TCP Preforked Server, Descriptor Passing

Preforking or Process Pool



Preforking or Process Pool

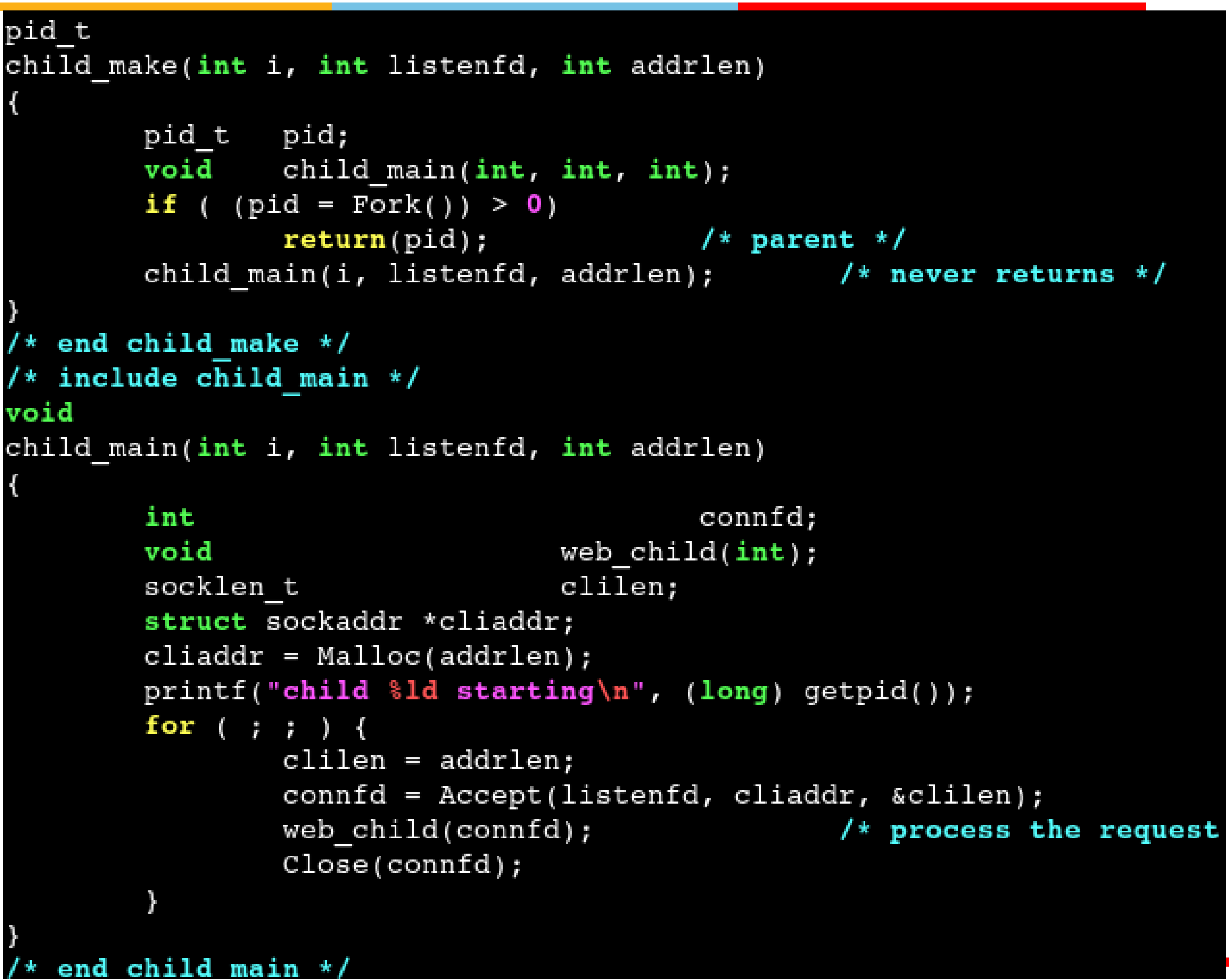


```
static int              nchildren;
static pid_t    *pids;

_
int
main(int argc, char **argv)
{
        int                     listenfd, i;
        socklen_t       addrlen;
        void            sig_int(int);
        pid_t           child_make(int, int, int);
        if (argc == 3)
                listenfd = Tcp_listen(NULL, argv[1], &addrlen);
        else if (argc == 4)
                listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
        else
                err_quit("usage: serv02 [ <host> ] <port#> <#children>");
        nchildren = atoi(argv[argc-1]);
        pids = Calloc(nchildren, sizeof(pid_t));
        for (i = 0; i < nchildren; i++)
                pids[i] = child_make(i, listenfd, addrlen);     /* parent returns */

        Signal(SIGINT, sig_int);
        for ( ; ; )
                pause();        /* everything done by children */
}
```

Preforking or Process Pool: No Locking Around Accept



```
pid_t
child_make(int i, int listenfd, int addrlen)
{
        pid_t   pid;
        void    child_main(int, int, int);
        if ( (pid = Fork()) > 0)
                return(pid);            /* parent */
        child_main(i, listenfd, addrlen);       /* never returns */
}
/* end child_make */
/* include child_main */
void
child_main(int i, int listenfd, int addrlen)
{
        int                             connfd;
        void                    web_child(int);
        socklen_t               clilen;
        struct sockaddr *cliaddr;
        cliaddr = Malloc(addrlen);
        printf("child %ld starting\n", (long) getpid());
        for ( ; ; ) {
                clilen = addrlen;
                connfd = Accept(listenfd, cliaddr, &clilen);
                web_child(connfd);              /* process the request
                Close(connfd);
        }
}
/* end child_main */
```

Preforking or Process Pool: No Locking Around Accept

- Advantages:
  - No cost of fork() before responding to client.
  - Process control is simpler.
- Disadvantages:
  - Parent must guess how many children to fork.
  - If too less, clients will experience delays in response.
  - If too excessive, system performance degrades.