COMP 8005
Project
Design

Harman Dhillon
A00994245
Dec 3, 2025

Purpose

- A distributed password-cracking system that demonstrates dynamic work distribution, checkpointing, worker failure recovery, and multi-threaded brute-force password cracking.

Data Types

Arguments

Server

Field	Type	Description
argc	integer	The number of arguments
argv	string[]	The arguments
program_name	string	The name of the program
Server IP	string	IP address of the server
Port number	integer	Port number
Hash	string	Hash to crack
Work size	integer	Number of passwords assigned per node request
checkpoint	integer	Number of attempts before a node sends a checkpoint
timeout	integer	Number of seconds to wait for a checkpoint from a client

Client

Field	Type	Description
argc	integer	The number of arguments
argv	string[]	The arguments
program_name	string	The name of the program
Server IP	string	IP address of the server

Port number	integer	Port to connect to
threads	integer	Number of threads to use for cracking

Context

Purpose: To hold the arguments, settings, and exit information

Field	Type	Description
arguments	Arguments	The command line arguments
settings	Settings	The parsed command line arguments
exit_code	integer	The exit code of the program
exit_message	string	The error message to print before exiting

Functions

Server

command_line.c

Function	Description
parse_arguments	Parse the command line arguments
handle_arguments	Verify the command line arguments
usage	Display a usage message when the command line argument has an issue
parse_in_port_t	Parses port string to port_t

fsm.c

Function	Description
fsm_run	Iterate through the states in the FSM
fsm_transition	Handle the transition between 2 states

server_config.c

Function	Description
socket_create	Creates a socket with the given domain, type, and protocol.
start_listening	Marks a socket as a listening socket with the specified backlog.
socket_accept_connection	Accepts an incoming client connection and returns the new socket descriptor.
socket_close	Closes the given socket descriptor safely.
socket_bind	Binds a socket to the provided sockaddr.
close_clients	Closes all active client sockets and frees their associated worker states.
handle_new_client	Handles accepting a new client and adding it to the client socket array.
get_sockaddr_info	Extracts human-readable IP and port strings from a sockaddr.
assign_work_to_client	Assigns the next unit of cracking work to a connected client.
process_client_message	Processes an incoming message from a client socket.
handle_single_message	Parses and handles a single protocol message from a client.
handle_client_disconnect	Cleans up after a client disconnects and compacts the client list.
reclaim_and_redistribute	Reclaims unfinished work from a disconnected client and redistributes it.
convert_address	Converts an IP string and port into a sockaddr_storage structure.
polling	Performs poll() on all client and server sockets and handles incoming events.

main.c

Function	Description
main	Main method to run the program.
parse_arguments_handler	Method to handle the STATE_PARSE_ARGUMENTS state

handle_arguments_handler	Method to handle the STATE_HANDLE_ARGUMENTS state
convert_address_handler	Method to handle the STATE_CONVERT_ADDRESS state
create_socket_handler	Method to handle the STATE_CREATE_SOCKET state
bind_socket_handler	Method to handle the STATE_BIND_SOCKET state
listen_handler	Method to handle the STATE_LISTEN state
setup_signal_handler	Method to handle the STATE_SETUP_SIGNAL state
start_timer_handler	Method to handle the STATE_START_TIMER state
start_polling_handler	Method to handle the STATE_START_POLLING state
stop_timer_handler	Method to handle the STATE_STOP_TIMER state
cleanup_handler	Method to handle the STATE_CLEANUP state
error_handler	Method to handle the STATE_ERROR state

States

State	Description
STATE_PARSE_ARGUMENTS	Parse command line arguments
STATE_HANDLE_ARGUMENTS	Verify and convert the command line arguments for use
STATE_CONVERT_ADDRESS	Convert ip address and port to sockaddr_storage
STATE_CREATE_SOCKET	Create socket
STATE_BIND_SOCKET	Bind socket
STATE_LISTEN	Start listening
STATE_SETUP_SIGNAL	Setup signal handlers
STATE_START_TIMER	Start the timer

STATE_START_POLLING	Start polling
STATE_STOP_TIMER	Stop the timer and calculate how long it took to perform the attack
STATE_ERROR	Display an error message
STATE_CLEANUP	Cleanup before exit

Client

command_line.c

Function	Description
parse_arguments	Parse the command line arguments
handle_arguments	Verify the command line arguments
usage	Display a usage message when the command line argument has an issue
parse_in_port_t	Parses port string to port_t
convert_to_int	Convert string to int

fsm.c

Function	Description
fsm_run	Iterate through the states in the FSM
fsm_transition	Handle the transition between 2 states

server_config.c

Function	Description
socket_create	Creates a socket with the given domain, type, and protocol.
socket_close	Closes the given socket descriptor safely.
socket_bind	Binds a socket to the provided sockaddr.
receive_hash	Wait for hash from the server
wait_for_work	Wait for work to be sent by server

send_checkpoint	Send checkpoint to server
send_done	Send done to server
send_found	Send found message with password
get_sockaddr_info	Extracts human-readable IP and port strings from a sockaddr.

cracker.c

Function	Description
index_to_password	Converts a numerical index to a string using the predefined charset
worker	Each thread tries candidate passwords and compares to the hash
create_threads	Creates threads that attempt to bruteforcely crack the hash

main.c

Function	Description
main	Main method to run the program.
parse_arguments_handler	Method to handle the STATE_PARSE_ARGUMENTS state
handle_arguments_handler	Method to handle the STATE_HANDLE_ARGUMENTS state
convert_address_handler	Method to handle the STATE_CONVERT_ADDRESS state
create_socket_handler	Method to handle the STATE_CREATE_SOCKET state
connect_socket_handler	Method to handle the STATE_CONNECT_SOCKET state
wait_hash_handler	Method to handle the STATE_WAIT_HASH state
wait_work_handler	Method to handle the STATE_WAIT_WORK state
start_timer_handler	Method to handle the STATE_START_TIMER state
start_cracking_handler	Method to handle the STATE_START_CRACKING state
send_done_handler	Method to handle the STATE_SEND_DONE state

stop_timer_handler	Method to handle the STATE_STOP_TIMER state
cleanup_handler	Method to handle the STATE_CLEANUP state
error_handler	Method to handle the STATE_ERROR state

States

State	Description
STATE_PARSE_ARGUMENTS	Parse command line arguments
STATE_HANDLE_ARGUMENTS	Verify and convert the command line arguments for use
STATE_CONVERT_ADDRESS	Convert ip address and port to sockaddr_storage
STATE_CREATE_SOCKET	Create socket
STATE_CONNECT_SOCKET	Connect to server
STATE_WAIT_HASH	Wait for hash from server
STATE_WAIT_WORK	Wait for work from server
STATE_START_TIMER	Start the timer
STATE_START_CRACKING	Start cracking
STATE_SEND_DONE	Send done to server
STATE_STOP_TIMER	Stop the timer and calculate how long it took to perform the attack
STATE_ERROR	Display an error message
STATE_CLEANUP	Cleanup before exit

State Table

Server

From State	To State	Function
FSM_INIT	STATE_PARSE_ARGUMENTS	parse_arguments_handler
STATE_PARSE_ARGUMENTS	STATE_HANDLE_ARGUMENTS	handle_arguments_handler
STATE_HANDLE_ARGUMENTS	STATE_CONVERT_ADDRESS	convert_address_handler
STATE_CONVERT_ADDRESS	STATE_CREATE_SOCKET	create_socket_handler
STATE_CREATE_SOCKET	STATE_BIND_SOCKET	bind_socket_handler
STATE_BIND_SOCKET	STATE_LISTEN	listen_handler
STATE_LISTEN	STATE_SETUP_SIGNAL	setup_signal_handler
STATE_SETUP_SIGNAL	STATE_START_TIMER	start_timer_handler
STATE_START_TIMER	STATE_START_POLLING	start_polling_handler
STATE_START_POLLING	STATE_STOP_TIMER	stop_timer_handler
STATE_STOP_TIMER	STATE_CLEANUP	error_handler
STATE_ERROR	STATE_CLEANUP	error_handler
STATE_PARSE_ARGUMENTS	STATE_ERROR	error_handler
STATE_HANDLE_ARGUMENTS	STATE_ERROR	error_handler
STATE_CONVERT_ADDRESS	STATE_ERROR	error_handler
STATE_CREATE_SOCKET	STATE_ERROR	error_handler
STATE_BIND_SOCKET	STATE_ERROR	error_handler

STATE_LISTEN	STATE_ERROR	error_handler
STATE_SETUP_SIGNAL	STATE_ERROR	error_handler
STATE_START_TIMER	STATE_ERROR	error_handler
STATE_START_POLLING	STATE_ERROR	error_handler
STATE_STOP_TIMER	STATE_ERROR	error_handler
STATE_CLEANUP	FSM_EXIT	

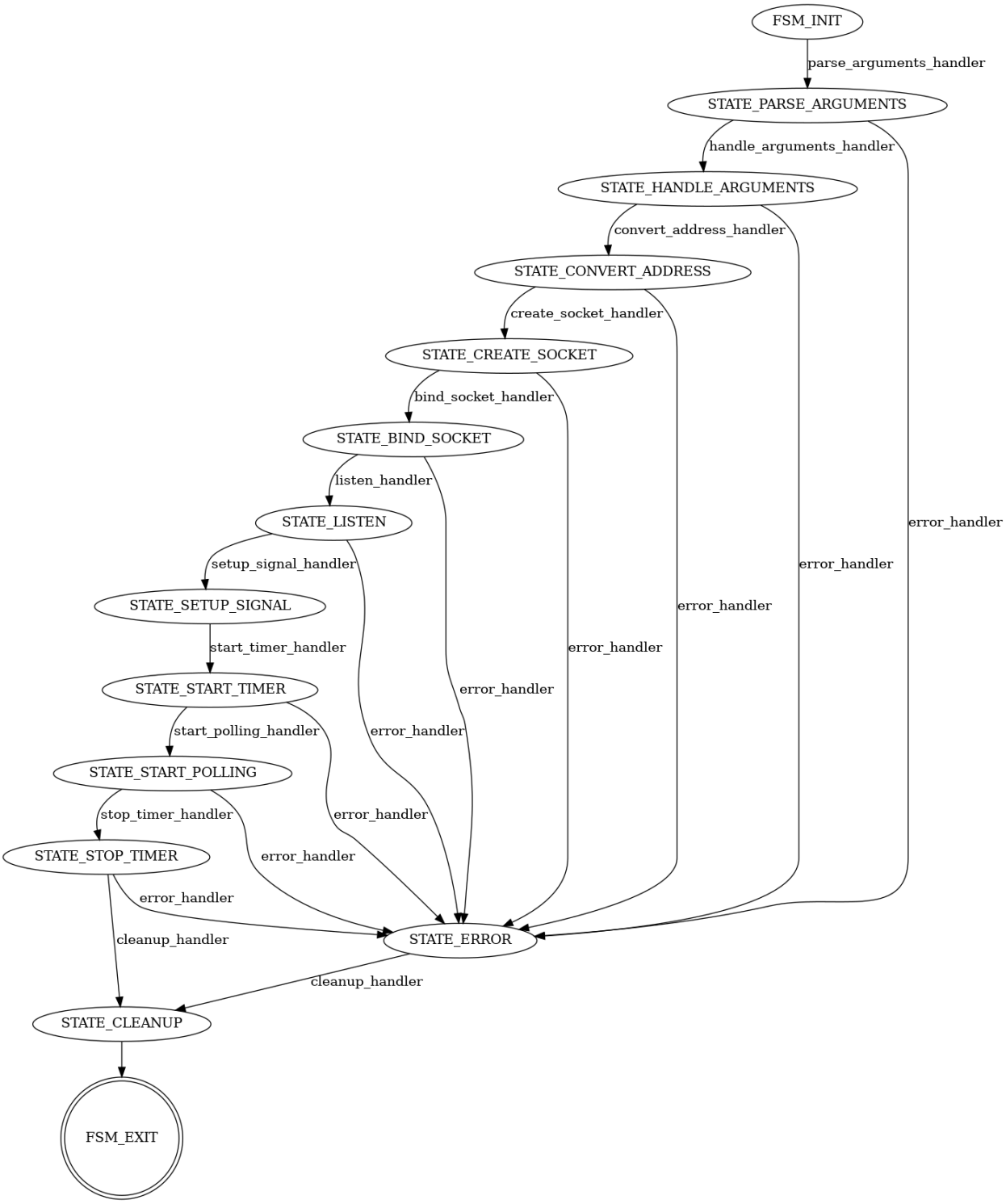
Client

From State	To State	Function
FSM_INIT	STATE_PARSE_ARGUMENTS	parse_arguments_handler
STATE_PARSE_ARGUMENTS	STATE_HANDLE_ARGUMENTS	handle_arguments_handler
STATE_HANDLE_ARGUMENTS	STATE_CONVERT_ADDRESS	convert_address_handler
STATE_CONVERT_ADDRESS	STATE_CREATE_SOCKET	create_socket_handler
STATE_CREATE_SOCKET	STATE_CONNECT_SOCKET	connect_socket_handler
STATE_CONNECT_SOCKET	STATE_WAIT_HASH	wait_hash_handler
STATE_WAIT_HASH	STATE_WAIT_WORK	wait_work_handler
STATE_WAIT_WORK	STATE_START_TIMER	start_timer_handler
STATE_WAIT_WORK	STATE_START_CRACKING	start_cracking_handler
STATE_WAIT_WORK	STATE_CLEANUP	cleanup_handler
STATE_START_TIMER	STATE_START_CRACKING	start_cracking_handler
STATE_START_CRAC	STATE_SEND_DONE	send_done_handler

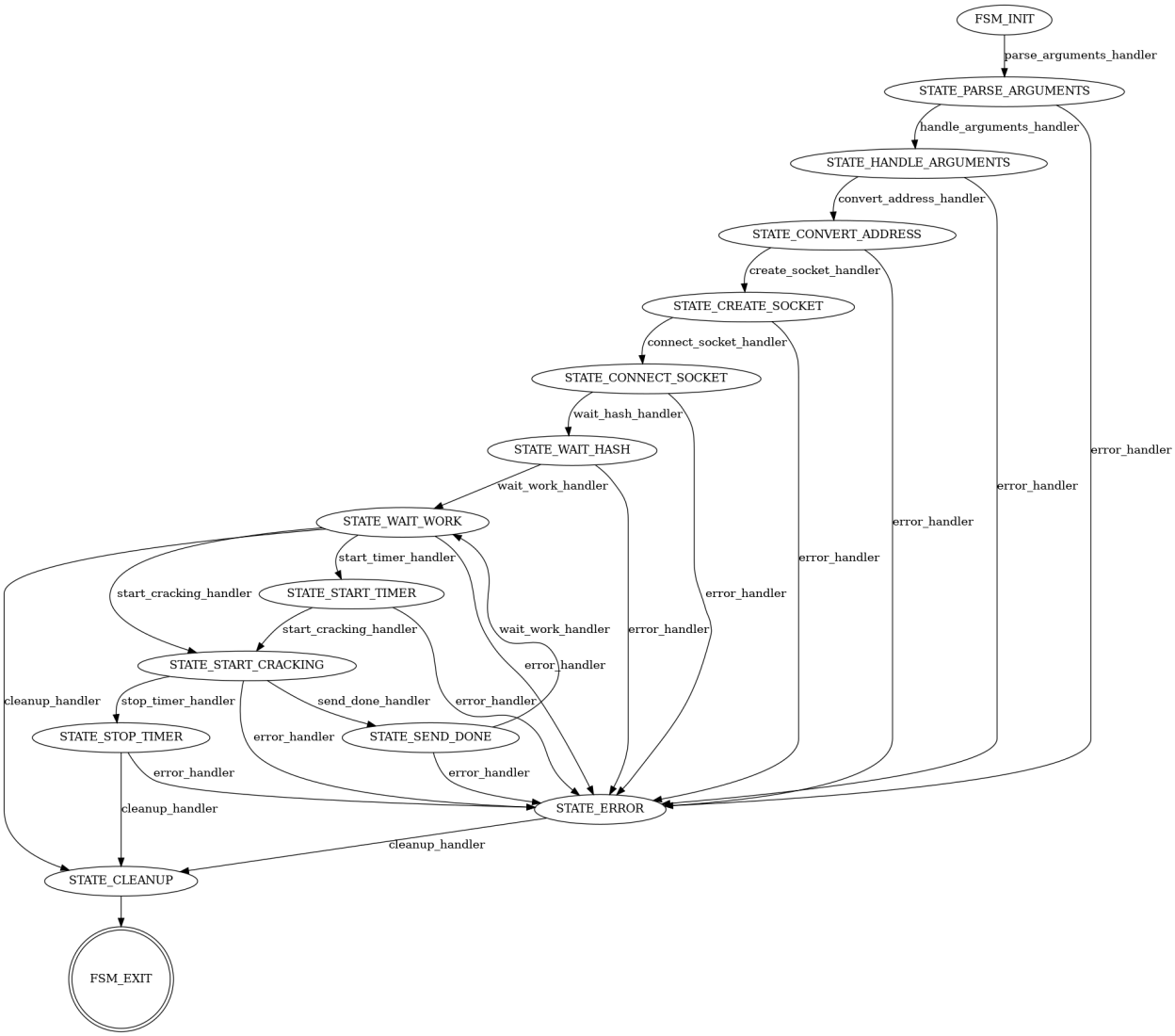
KING		
STATE_START_CRACKING	STATE_STOP_TIMER	stop_timer_handler
STATE_SEND_DONE	STATE_WAIT_WORK	wait_work_handler
STATE_STOP_TIMER	STATE_CLEANUP	error_handler
STATE_ERROR	STATE_CLEANUP	error_handler
STATE_PARSE_ARGUMENTS	STATE_ERROR	error_handler
STATE_HANDLE_ARGUMENTS	STATE_ERROR	error_handler
STATE_CONVERT_ADDRESS	STATE_ERROR	error_handler
STATE_CREATE_SOCKET	STATE_ERROR	error_handler
STATE_CONNECT_SOCKET	STATE_ERROR	error_handler
STATE_WAIT_HASH	STATE_ERROR	error_handler
STATE_WAIT_WORK	STATE_ERROR	error_handler
STATE_WAIT_WORK	STATE_ERROR	error_handler
STATE_WAIT_WORK	STATE_ERROR	error_handler
STATE_START_TIMER	STATE_ERROR	error_handler
STATE_START_CRACKING	STATE_ERROR	error_handler
STATE_START_CRACKING	STATE_ERROR	error_handler
STATE_SEND_DONE	STATE_ERROR	error_handler
STATE_STOP_TIMER	STATE_ERROR	error_handler
STATE_CLEANUP	FSM_EXIT	

State Transition Diagram

Server


FSM_INIT
parse_arguments_handler
STATE_PARSE_ARGUMENTS
handle_arguments_handler
STATE_HANDLE_ARGUMENTS
convert_address_handler
STATE_CONVERT_ADDRESS
create_socket_handler
STATE_CREATE_SOCKET
bind_socket_handler
STATE_BIND_SOCKET
listen_handler
STATE_LISTEN
setup_signal_handler
STATE_SETUP_SIGNAL
start_timer_handler
STATE_START_TIMER
start_polling_handler
STATE_START_POLLING
stop_timer_handler
STATE_STOP_TIMER
cleanup_handler
STATE_CLEANUP
FSM_EXIT
STATE_ERROR
error_handler
cleanup_handler

Client


FSM_INIT
parse_arguments_handler
STATE_PARSE_ARGUMENTS
handle_arguments_handler
STATE_HANDLE_ARGUMENTS
convert_address_handler
STATE_CONVERT_ADDRESS
create_socket_handler
STATE_CREATE_SOCKET
connect_socket_handler
STATE_CONNECT_SOCKET
wait_hash_handler
STATE_WAIT_HASH
wait_work_handler
STATE_WAIT_WORK
start_timer_handler
start_cracking_handler
STATE_START_TIMER
start_cracking_handler
wait_work_handler
STATE_START_CRACKING
cleanup_handler
stop_timer_handler
send_done_handler
error_handler
STATE_STOP_TIMER
error_handler
STATE_SEND_DONE
error_handler
error_handler
cleanup_handler
STATE_ERROR
cleanup_handler
STATE_CLEANUP
FSM_EXIT
error_handler
error_handler
error_handler
error_handler
error_handler
error_handler

Pseudocode

Server

command_line.h

parse_arguments

Parameters

Parameter	Type	Description
argc	integer	The number of arguments passed to the program.
argv	Char array	The arguments passed to the program.
arguments	struct	Struct holding the arguments
err	struct fsm_error	To store errors

Return

Value	Reason
0	Arguments parsed successfully
-1	There were invalid arguments passed

Pseudo Code

```
call getopt in a loop
     if opt is ‘h’
          call usage
     if opt is ‘p’
          Set port to value from flag
     if opt is ‘s’
          Set server to value from flag
     if opt is ‘t’
          Set thread to value from flag
     If opt is ‘?’
          call usage
          Return -1

If there are more than 1 unnamed argument
```

```
      return -1

if f or u flag not passed
      Return -1

return 0
```

handle_arguments

Parameters

Parameter	Type	Description
threads	int	Number of threads

Return

Value	Reason
0	Finished function

Pseudo Code

```
if threads is not set
      Set threads to 1

return 0
```

usage

Parameters

Parameter	Type	Description
program_name	Const char *	Program name

Return

void

Pseudo Code

```
print the program name and how to run the program to stderr
```

server_config.c

polling

Parameters

Parameter	Type	Description
sockfd	Struct fsm_context	runtime context
file_descriptors	Struct pollfd	Holds file descriptors of the server
max_clients	nfds_t	Max clients connected
client_sockets	Int array	Array of socket fds
client_states	Struct worker_state	Holds the state of each worker
crack_ctx	Struct cracking_context	Holds the cracking context
err	Struct fsm_error	for error reporting

Return

Value	Reason
0	No error
-1	error occurred

Pseudo Code

```
Do
     Set temp_fds to realloc file_descriptors for (max_clients + 2)
pollfd entries
     If realloc fails
          Set error message
          Return -1

     Update file_descriptors pointer to temp_fds

     Set file_descriptors[0] to monitor sockfd for POLLIN
```

```
     For each client index i from 0 to max_clients - 1
          Retrieve the client's socket fd
          Set file_descriptors[i + 1] to monitor this fd for POLLIN
          Update client_states[i].sockfd to this fd
     EndFor

     Call poll() on the file_descriptors array with a timeout of
1000 ms

     If poll returned an error
          If error was EINTR
               Return 0
          Set error message
          Return -1

     If the listening socket (index 0) has POLLIN
          Call handle_new_client to accept a new connection
          If a new client was accepted
               Resize the client_states array
               Allocate a new worker_state for the new client
               Initialize worker_state fields (sockfd, alive,
assigned, last_heard, recv_len)
               Send hash to the worker
               Decrement num_ready
          EndIf
     EndIf

     For each client index i from 0 to max_clients - 1
          Retrieve worker_state ws for this client
          If worker is not alive
               Continue to next client

          If this client's pollfd has POLLIN
               Retrieve client socket
               If process_client_message returns -1
                    If cracking not complete
                         Reclaim and redistribute the client's work
                    Disconnect client
                    Continue to next client
               EndIf
               Update ws.last_heard
               Decrement num_ready
          EndIf
```

```
            If (current time - ws.last_heard) > ws.timeout_seconds
                  Print timeout message
                  Reclaim and redistribute work
                  Disconnect client
            EndIf
      EndFor


Return 0
```

assign_work_to_client

Parameters

Parameter	Type	Description
client_states	Struct worker_state	Holds the state of each worker
crack_ctx	Struct cracking_context	Holds the cracking context
err	Struct fsm_error	for error reporting

Return

Value	Reason
0	No error
-1	error

Pseudo Code

```
Do
      If cracking_context indicates the password is already found
            Send "STOP" message to this worker
            Return 1
      EndIf

      Set start and len to zero
      Call pop_next_work_chunk to retrieve the next work range
```

```
      Update worker_state fields:
            Set start_index to start
            Set work_size to len
            Set end_index to (start + len - 1)
            Set last_checkpoint_index to start
            Set assigned to true
            Set started_at to current time
            Set last_heard to started_at
            Set checkpoint_interval from cracking_context
            Set timeout_seconds from cracking_context

      Format a WORK message into a buffer with:
            start_index, work_size, checkpoint_interval,
timeout_seconds

      Send the WORK message to the worker

      If send failed
            Set error message
            Return -1
      EndIf

      Print server log that the worker received work



Return 0
```

handle_single_message

Parameters

Parameter	Type	Description
sd	int	Socket descriptor
client_states	Struct worker_state	Holds the state of each worker
crack_ctx	Struct cracking_context	Holds the cracking context

buffer	Char *	Current message
err	Struct fsm_error	for error reporting

Return

Value	**Reason**
0	No error
-1	error
1	Found password

Pseudo Code

```
Do
      If message begins with "READY"
            Print that the worker is READY
            If cracking is not yet found
                  Call assign_work_to_client
                  If assign_work_to_client failed
                        Return -1
            EndIf
            Return 0
      EndIf

      If message begins with "CHECKPOINT "
            Parse checkpoint index from the message
            If checkpoint index is outside worker’s assigned range
                  Set error message
                  Return -1
            EndIf

            Set now to current time
            Add (now - last_heard) to crack_ctx.total_secs
            Update worker's last_checkpoint_index
            Update worker's last_heard to now

            Print checkpoint information
            Return 0
      EndIf

      If message begins with "FOUND "
```

```
            Extract the password from the message
            Set now to current time
            Add (now - last_heard) to crack_ctx.total_secs

            Print that worker found the password and time taken

      EndIf

Return appropriate value based on branch taken
```

pop_next_work_chunk

Parameters

Parameter	Type	Description
crack_ctx	Struct cracking _context	Holds the cracking context
out_start	uint64_t	Starting index
out_len	uint64_t	Work size

Return

Value	Reason
1	No error

Pseudo Code

```
Do
      If the work queue contains one or more queued chunks
            Set last to (queue_len - 1)

            Set out_start to queue[last].start
            Set out_len   to queue[last].len

            Decrement queue_len

            If queue_len becomes zero
                  Free the queue
                  Set queue pointer to NULL
            Else
```

```
            Shrink the queue with realloc to its new length
        EndIf

        Return true
    EndIf

    Set out_start to ctx.index
    Set out_len   to ctx.work_size

    Increment ctx.index by ctx.work_size

    Return true
```

fsm.c

fsm_run

Parameters

Parameter	Type	Description
context	Struct fsm_context	runtime context
err	Struct fsm_error	for error reporting
from_state	int	optional out param (could store last from state)
to_state	int	optional out param (could store last to state)
transitions	const struct client_fsm_transition	Transition table array

Return

Value	Reason
0	FSM completed normally (reached FSM_EXIT)

-1	error occurred (missing transition or transition handler error)

Pseudo Code

```
Do
      Set initialize transition from FSM_INIT to FSM_USER_START
      Get the next transition
      If no transition exists
            Return -1
      Call perform
While to_id does not equal FSM_EXIT
Return 0
```

fsm_transition

Parameters

Parameter	Type	Description
context	Struct fsm_context	runtime context
err	Struct fsm_error	for error reporting
from_id	int	current-from state id
to_id	int	desired-to state id
transitions	const struct client_fsm_transition	Transition table array

Return

Value	**Reason**
fsm_state_func	The next transition
NULL	No matching transition

Pseudo Code

```
Iterate over the transitions array until you find the matching
from_id and to_id
      Return transition
```

```
Return NULL
```

Client

command_line.h

parse_arguments

Parameters

Parameter	Type	Description
argc	integer	The number of arguments passed to the program.
argv	Char array	The arguments passed to the program.
arguments	struct	Struct holding the arguments
err	struct fsm_error	To store errors

Return

Value	Reason
0	Arguments parsed successfully
-1	There were invalid arguments passed

Pseudo Code

```
call getopt in a loop
     if opt is ‘h’
          call usage
     if opt is ‘p’
          Set port to value from flag
     if opt is ‘s’
          Set server to value from flag
     if opt is ‘t’
          Set thread to value from flag
     If opt is ‘?’
          call usage
          Return -1

If there are more than 1 unnamed argument
```

```
      return -1

if f or u flag not passed
      Return -1

return 0
```

handle_arguments

Parameters

Parameter	Type	Description
threads	int	Number of threads

Return

Value	Reason
0	Finished function

Pseudo Code

```
if threads is not set
      Set threads to 1

return 0
```

usage

Parameters

Parameter	Type	Description
program_name	Const char *	Program name

Return

void

Pseudo Code

```
print the program name and how to run the program to stderr
```

fsm.c

fsm_run

Parameters

Parameter	Type	Description
context	Struct fsm_context	runtime context
err	Struct fsm_error	for error reporting
from_state	int	optional out param (could store last from state)
to_state	int	optional out param (could store last to state)
transitions	const struct client_fsm_transition	Transition table array

Return

Value	Reason
0	FSM completed normally (reached FSM_EXIT)
-1	error occurred (missing transition or transition handler error)

Pseudo Code

```
Do
      Set initialize transition from FSM_INIT to FSM_USER_START
      Get the next transition
      If no transition exists
            Return -1
      Call perform
While to_id does not equal FSM_EXIT
Return 0
```

fsm_transition

Parameters

Parameter	Type	Description
context	Struct fsm_context	runtime context
err	Struct fsm_error	for error reporting
from_id	int	current-from state id
to_id	int	desired-to state id
transitions	const struct client_fsm_transition	Transition table array

Return

Value	Reason
fsm_state_func	The next transition
NULL	No matching transition

Pseudo Code

```
Iterate over the transitions array until you find the matching
from_id and to_id
     Return transition
Return NULL
```

cracker.c

index_to_password

Parameters

Parameter	Type	Description
index	int	Numeric index to convert to password

Return

Value	Reason
Char *	Pointer to malloc'd string containing the password

Pseudo Code

```
Set len to 1
Set range to the size of the password character charset size
Set total to range

While index greater than or equal to total
      Set len to len + 1
      Set range to range * charset_size
      Set total to total + range

Set start_index to total - range
Set start_index to index - start_index

For pos from len - 1 to 0
      Set digit to local_index % charset_size
      Set buffer[pos] to charset[digit]
      Set local_index to local_index / charset_size

Return buffer
```

worker(thread function)

Parameters

Parameter	Type	Description
arg	void*	Pointer to hash string to match

Return

Pseudo Code

```
While password not found by any other thread
      If index > worksize - 1
            break
      If index mod checkpoint
            Send checkpoint
      Call index_to_password
```

```
      Encrypt the password
      Compare password hash to the hash passed in
      If hashes match
            Set found password atomic to true
            Turn on the found mutex
            Set the found_candidate to the password
            Unlock the mutex

Return NULL
```

create_threads

Parameters

Parameter	Type	Description
number_of_Threads	int	number of worker threads to create
hash	Const char*	target hash string

Return

Value	Reason
0	success (password found)
-1	failure (thread creation error or password not found)

Pseudo Code

```
For loop for the amount of threads to create
      Create worker threads
Return 0 if password found or -1 if error or no password found
```