**Check how the death of a philosopher is checked and if there is a mutex to protect that a philosopher dies and start eating at the same time.**

The death of a philosopher is checked in the function [ft\_break\_while](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/)) in [main.c](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/). The function checks if the time since the last meal exceeds the [time2die](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/) or if all philosophers have finished eating the required number of meals.

Here is the relevant code excerpt from [main.c](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/):

bool    ft\_break\_while(t\_args \*args, int \*i)

{

    pthread\_mutex\_lock(&args->sync\_mutex); // Lock the sync\_mutex at the beginning

    if (ft\_now\_ms() - args->philos[\*i].last\_meal\_beginning

        >= (unsigned long long)args->time2die

        || args->total\_finished == args->philo\_count)

    {

        if (args->max\_meals != -1 && args->total\_finished == args->philo\_count)

        {

            args->everybody\_full = true;

            printf("Every philosopher has eaten %d times\n", args->max\_meals);

        }

        else

        {

            args->death\_occured = true;

            printf("%lu %d died\n",

                ft\_now\_ms() - args->start\_time, args->philos[\*i].nbr);

        }

        pthread\_mutex\_unlock(&args->sync\_mutex); // Ensure sync\_mutex is unlocked before returning

        return (true);

    }

    if (\*i + 1 == args->philo\_count)

        \*i = -1;

    pthread\_mutex\_unlock(&args->sync\_mutex); // Ensure sync\_mutex is unlocked before returning

    return (false);

}

The [sync\_mutex](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/) is used to protect the shared state variables [death\_occured](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/) and [everybody\_full](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/) from concurrent access. This ensures that the state is consistently updated and read.

In the philosopher's routine function [proutine](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/)) in [philo\_utils.c](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/), the [sync\_mutex](https://vigilant-space-enigma-69966qqxpp59f497g.github.dev/) is also used to protect the philosopher's state when they start eating:

void    \*proutine(void \*data)

{

    t\_philo \*philo;

    philo = (t\_philo \*)data;

    if (!(philo->nbr & 1))

        ft\_usleep(10);

    while (1)

    {

        if (!ft\_get\_forks(philo))

            break ;

        pthread\_mutex\_lock(&philo->args->sync\_mutex);

        philo->meals\_count++;

        philo->last\_meal\_beginning = ft\_now\_ms();

        pthread\_mutex\_unlock(&philo->args->sync\_mutex);

        ft\_write\_status(philo, "is eating");

        ft\_usleep(philo->args->time2eat);

        pthread\_mutex\_lock(&philo->args->sync\_mutex);

        if (philo->meals\_count == philo->args->max\_meals)

            philo->args->total\_finished += (philo->args->max\_meals != -1);

        pthread\_mutex\_unlock(&philo->args->sync\_mutex);

        pthread\_mutex\_unlock(&philo->l\_fork);

        pthread\_mutex\_unlock(philo->r\_fork);

        if (!ft\_sleep\_think(philo))

            break ;

    }

    return (NULL);

}

**Philosophers Bonus!!!**

#include <semaphore.h>

#include <sys/types.h>

#include <unistd.h>

// Define a semaphore for forks

sem\_t forks;

int main(int argc, char \*\*argv)

{

t\_args args;

int i;

memset(&args, 0, sizeof(t\_args));

if (argc < 5 || argc > 6 || !ft\_initialize\_args(&args, argv))

return (write(2, "Error: Invalid number arguments\n", 30), 1);

// Initialize the semaphore with the number of forks

sem\_init(&forks, 0, args.philo\_count);

ft\_initialize\_philos(&args);

for (i = 0; i < args.philo\_count; i++)

{

pid\_t pid = fork();

if (pid == 0)

{

// Child process (philosopher)

proutine(&args.philos[i]);

exit(0);

}

}

// Main process waits for all philosophers to finish

for (i = 0; i < args.philo\_count; i++)

wait(NULL);

// Clean up

sem\_destroy(&forks);

return 0;

}