Yes, you can in fact. For a deadlock to occur you may have the following preconditions:

**Mutual Exclusion**: processes hold a resource exclusively.

**Non preemption**: Only the process itself can release the resource it is holding

**Resource waiting**: Process requests unavailable resource, and block title they become available

**Partial Allocation**: Processes may hold some of the limited resource and request additional units of the same or different resource.

Only one of these can apply to a single process **Resource waiting**. The rest would require a system of processes.

One scenario is with pipes where you create a pipe and have both the read and write end open but then instead of writing to the pipe you read. Read is a waiting call that will try to read from the pipe as long as the write end is open and/or it hasn’t completely read the number of bytes it was asked to read.

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#define size 100

#define READ 0

#define WRITE 1

#define SET 2

//MAIN//

int main(){

int fd[SET]; //pipe file descriptor set

char message[size]; //message container to read into

pipe(fd); //create unneamed pipe

printf("begin reading\n");

/\*deadlock state\*/

int bytered = read(fd[0], message, size);

close(fd[READ]), close(fd[WRITE]); //close pipe

printf("%s\n", message);//print message red

printf("end read(will never reach here)\n");

exit(0);

}//END MAIN

here essentially you’re creating a pipe to yourself where you are waiting for bytes that were never written.

Or one with semaphore where you initialize the semaphore to zero and wait on it.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

//SEMAPHORE

sem\_t deadsem;

//MAIN//

int main(){

sem\_init(&deadsem, 0, 0);

printf("deadlock semaphore\n");

sem\_wait( &sem ); //deadlock condition

printf("After wait (will never reach here)");

exit(0);

}//END MAIN