NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

MA4830 Major Project

Group members:

1) Iyer Venkataraman Natarajan	**U1822259F**
2) Mohammed Adnan Azam	**U1822486K**
3) Sarvesh Tusnial	**U1722677A**
4) Chan Wei Jian	**U1822389H**
5) Wu Po Yi	**U1822557D**
6) Ang Yu Pin	**U1822354D**

Group Photo:

1. <u>Introduction</u>

A **metronome** is a device that produces an audible click or other sound at a regular interval that can be set by the user, typically in beats per minute (BPM). Musicians use the device to practice playing to a regular pulse. Metronomes typically include synchronized visual motion (e.g., swinging pendulum or blinking lights).

Our program aims to create a digital version of a metronome, also known as a “software metronome” that runs on QNX Neutrino RTOS, which aims to be both accurate and easy to use. Our program must be capable of accepting the user’s input and adjust the metronome’s BPM in real time. A design visual accompanies the sound to enhance the user’s experience in using the digital metronome.

2. How does it work?

Our program is designed to accept an input of BPM from the user, which triggers the start of the program. We achieved our design with a total of **3 threads**, which would run simultaneously:

1) **Main thread** for the running of the program
2) **Metronome thread** to generate the metronome beat
3) **Keyboard thread** for receiving input from the user via the keyboard.

2.1 Main Thread

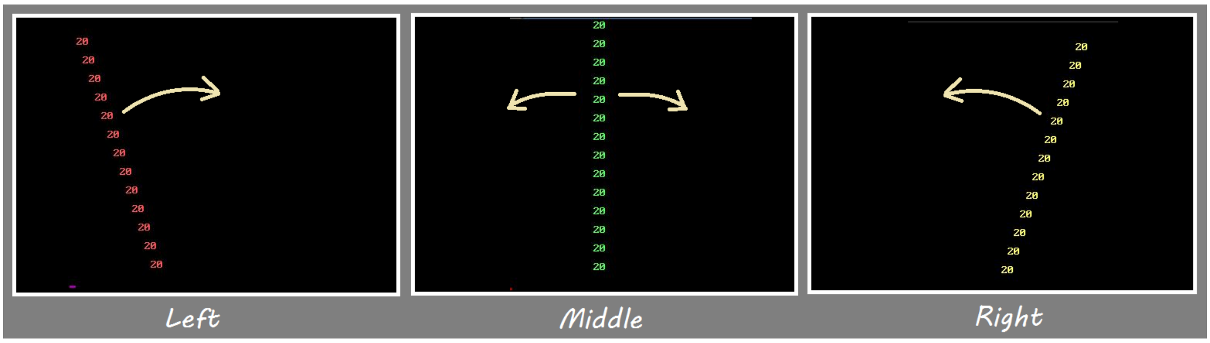
Firstly, an initial BPM input is required by the user together with the command to start the program. The BPM input would be stored as a global variable. Upon starting, the main thread triggers the start of the Metronome and Keyboard threads. Instructions to manipulate the metronome BPM will be displayed.

2.2 Metronome Thread

To generate a regular beat, we made use of a clock function within QNX. With the first BPM input of the user, and the interval of each beat is calculated and read by the metronome thread. This produces the initial regular beat of the program.

The input beats per minute is read in as the second argument (./ met 50, 50 BPM in this case) when running the code. The code then calculates the time taken for the number of beats per pattern to be 0.25 times the time taken for one beat to take place. This time taken for the number of beats per pattern is then stored in timer attribute variable itime.it_interval.tv_sec.

As the metronome thread runs, a visual display of a pixelated metronome is animated by printing characters in lines and diagonals. The animation loops around the 3 different positions: left, middle and right, while changing colours for every beat. This allows the user to have a visual appreciation of the digital metronome, on top of the sound generated. The animated metronome is timed to the BPM and the BPM is displayed as the individual unit that made up the lines and diagonals.



2.3 Keyboard Thread

The keyboard thread is simultaneously running in the background, to detect any input from the keyboard. This allows the user to control the BPM of the metronome in real time with the keyboard. Our program only detects input from the up and down arrow keys. The change is quickly registered and sent to the metronome thread, where the timer object is reset with the updated parameters, which reacts by increasing or decreasing the BPM by a multiplier of 1.2 respectively.

The metronome continues running until it is terminated with Ctrl+C by the user.

3. Novelty

- There is no upper or lower limit in the programmable beats per minute.
- Prompt the users to re-input correct values if the users have entered wrong input. E.g. entered alphabets instead of numbers.

4. Limitation

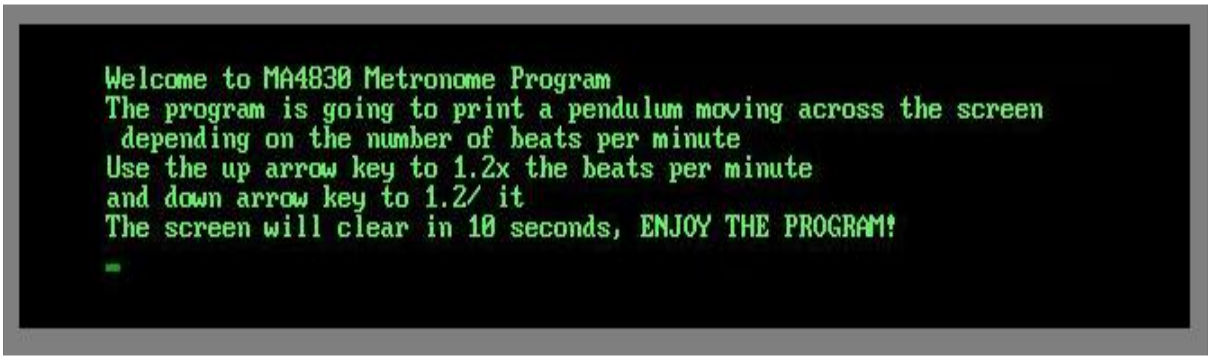
- The highest BPM the metronome is capable of clearly outputting and is limited by the default length of each beep. At very high (>1200) BPM, the interval between each beat would be so short that there would be an overlap of sound. Thus, it is impossible to differentiate the individual beat from each other.
- At very high frequency, aliasing occurs and the visualisation of the swinging metronome glitches.
- Upon the initial input of the metronome beats per minute, the beats could only increased or decreased by a fixed multiplier of 1.2

5. Instructions for User

To execute the digital metronome, include the desired BPM on the same ./ line, separated by a space. An example is shown below:

```
cc -o met metronome_final.c
./ met 50
```

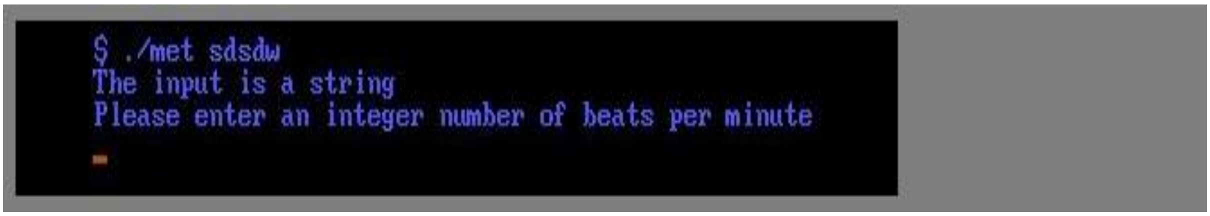
This would start the digital metronome program at 50 BPM, running continuously until any input by the user is detected. Upon successful start, the user should see a new set of instructions printed:



```
Welcome to MA4830 Metronome Program
The program is going to print a pendulum moving across the screen
 depending on the number of beats per minute
Use the up arrow key to 1.2x the beats per minute
and down arrow key to 1.2/ it
The screen will clear in 10 seconds, ENJOY THE PROGRAM!
_
```

Fig 5.1 : Welcome message after valid input

If the input is incorrect, e.g., a non-integer was used for the BPM, the program would prompt the user to re-enter a correct input:

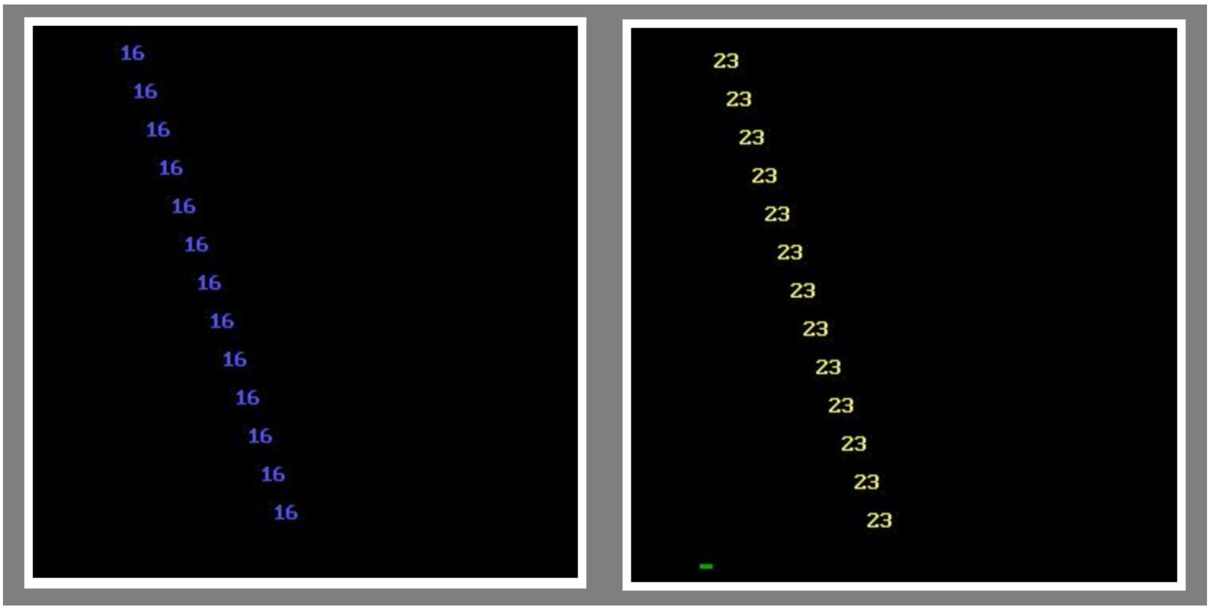


```
$ ./met sdsdw
The input is a string
Please enter an integer number of beats per minute
```

Fig 5.2 : Error message after invalid input

During runtime of the program, it accepts the following input by the user to control the BPM of the metronome:

- Pressing the **Up Arrow Key** would increase the BPM by a multiplier of 1.2 for each press
- Pressing the **Down Arrow Key** would decrease the BPM by a multiplier of 1.2 for each press

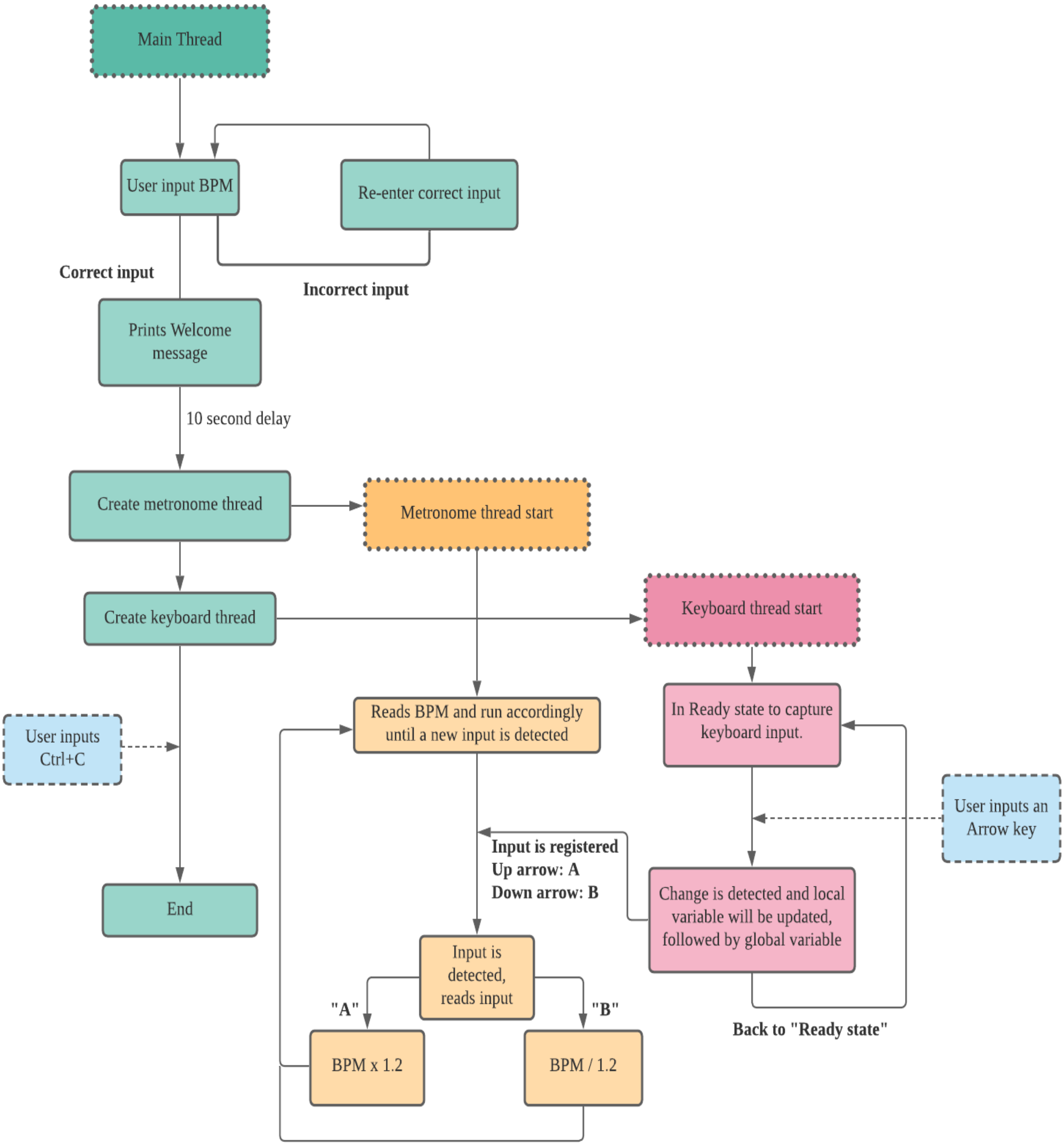


```
16
 16
  16
   16
    16
     16
      16
       16
        16
         16
          16
           16
            16
```

```
23
 23
  23
   23
    23
     23
      23
       23
        23
         23
          23
           23
            23
```

Pressing **Ctrl + C** would terminate the metronome beats.

6. Flowchart



The flowchart represents the general sequence of threads and processes that took place during the program's run time. The 3 threads are represented by a colour each: Main thread (Green), Metronome thread (Orange) and Keyboard thread (Pink). User's inputs are represented by blue boxes.

Appendix:

```c
/*Metronome program printing an oscillating pendulum with ability to increase and decrease the beats per minute using keyboard real-time input */

// Import all the required libraries
#include <stdio.h>

#include <time.h>

#include <sys/netmgr.h>

#include <sys/neutrino.h>

#include <stdlib.h>

#include <sys/iofunc.h>

#include <sys/dispatch.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#include <unistd.h>

#include <pthread.h>

#include <sched.h>

#include <math.h>

// Define various variables
#define PULSE_CODE _PULSE_CODE_MINAVAIL
#define PAUSE_CODE(PULSE_CODE + 1)
#define QUIT_CODE(PAUSE_CODE + 1)
#define ATTACH_POINT "metronome"
// Define cursor movement
#define cursup "\033[A"
#define cursdown "\033[B"
#define home "\033[0;0H"
// Define pendulum positions
#define initial_position "               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d\n\n               %d"
#define left_position " \n\n %d\n\n  %d\n\n   %d\n\n    %d\n\n     %d\n\n      %d\n\n       %d\n\n        %d\n\n          %d\n\n           %d\n\n            %d\n\n             %d\n\n              %d\n\n               %d"
#define right_position "                                \n\n                               %d\n\n                              %d\n\n                            %d\n\n                           %d\n\n                           %d\n\n                         %d\n\n                        %d\n\n                       %d\n\n                      %d\n\n                     %d\n\n                    %d\n\n                   %d\n\n                  %d\n\n                 %d"
```

```c
// Declare the global variables
int beatPerMin = 0;

// Variable to change pendulum color
char esc_code = 0x9B;

// Attach metronome to Event object
name_attach_t * attach;

// Union defining Message
typedef union {
  struct _pulse pulse;
  char msg[255];
}
my_message_t;

// Initialise mutual exclusion object
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
//macro

// Variable to store the keyboard arrow input and current beats per minute
typedef struct {
  int beatPerMin;
  char keyboard_inp;
}
input;

// Initialise it to dummy values
input inputs = {
  0,
  '\0'
};

// Variable to store condition if there is an input from the keyboard
int input_change = 0;

// Array to store pendulum positions for one oscillation
char pendulum_positions[4][500] = {
  initial_position,
  left_position,
  initial_position,
  right_position
};

/*Metronome function that runs the timer and pendulum on screen */
void * MetronomeFunc(void * arg) {

  // Declare the event and timer and message objects
  struct sigevent event;
  struct itimerspec itime;
  timer_t timer_id;
  int rcvid, index, i;
  my_message_t msg;

  // Variables to store the number of seconds per beat
```

```
double beat, fractional, beat_per_pattern;

// Iteration number to change the pendulum positions
int iter_number = 0;

// Color number to loop through different colors
int color_number = 1;

// Initialise the Event object
event.sigev_notify = SIGEV_PULSE;
event.sigev_coid = ConnectAttach(ND_LOCAL_NODE, 0, attach -> chid, _NTO_SIDE_CHANNEL, 0);
event.sigev_priority = SchedGet(0, 0, NULL);
event.sigev_code = PULSE_CODE;

// Create the timer
timer_create(CLOCK_REALTIME, & event, & timer_id);

// Get the Beats per minute from the global variable
beatPerMin = inputs.beatPerMin;

// The metronome would output 120 beats per minute (60 sec / 120 beats = 0.5 sec / beat).
beat = (double) 60 / beatPerMin;
beat_per_pattern = beat * (0.25);
fractional = beat_per_pattern - (int) beat_per_pattern;

// Initialise the timer starting and interval values
itime.it_value.tv_sec = 1;
itime.it_value.tv_nsec = 500000000;

itime.it_interval.tv_sec = beat_per_pattern;
itime.it_interval.tv_nsec = (fractional * 1e+9);

// Set the timer attributes
timer_settime(timer_id, 0, & itime, NULL);

for (;;) {

  if (iter_number == 4) {
    iter_number = 0;
  }

  if (color_number == 6) {
    color_number = 1;
  }

  // Check is there is an arrow key input, if yes, reset the timer attributes with the new Beats per minute
  if (input_change == 1) {

    // 1.2x the Beats per minute if up arrow pressed
    if (inputs.keyboard_inp == 'A') {
      inputs.beatPerMin *= 1.2;
    }

    // /1.2 the Beats per minute if down arrow pressed
    else if (inputs.keyboard_inp == 'B') {
      inputs.beatPerMin /= 1.2;
    }

    beatPerMin = inputs.beatPerMin;

    // Calculate the number of seconds per beat
    beat = (double) 60 / beatPerMin;

    // Calculate the number of seconds per pattern oscillation
    beat_per_pattern = beat * (0.25);
    fractional = beat_per_pattern - (int) beat_per_pattern;

    // Reset timer with new attributes
    itime.it_interval.tv_sec = beat_per_pattern;
    itime.it_interval.tv_nsec = (fractional * 1e+9);

    timer_settime(timer_id, 0, & itime, & itime);

    // Update the input variable
    input_change = 0;
  }

  while (1) {

    // check for a pulse from the user (pause, info or quit)
    rcvid = MsgReceive(attach -> chid, & msg, sizeof(msg), NULL);
    if (rcvid == 0) {
      if (msg.pulse.code == PULSE_CODE) {
        // Clear the screen
        putchar(12);

        // Print the position of pendulum in a color
        printf("%c1m%c=%dF", esc_code, esc_code, color_number);
        printf(pendulum_positions[iter_number], beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin, beatPerMin);

        // Move cursor to Home position
        printf(home);

        // Produce a beat
        printf("%c", 0x07);

        // Update the color and iteration numbers
        color_number++;
        iter_number++;

        break;
      } else if (msg.pulse.code == PAUSE_CODE) {
        itime.it_value.tv_sec = msg.pulse.value.sival_int;
        timer_settime(timer_id, 0, & itime, NULL);
      } else if (msg.pulse.code == QUIT_CODE) {
        printf("\nQuiting\n");

        TimerDestroy(timer_id);
```

```c
          exit(EXIT_SUCCESS);
        }
      }

      fflush(stdout);
    }
  }
  return EXIT_SUCCESS;
}

/*Function to capture input from Keyboard, the second thread */
void * KeyboardFunc(void * arg) {

  char keyboard_inp_local;

  for (;;) {

    while (1) {

      // Wait for input from user
      scanf("%c", & keyboard_inp_local);

      // Update the input_change variable and keyboard_inp in inputs structure
      pthread_mutex_lock( & mutex);
      input_change = 1;
      inputs.keyboard_inp = keyboard_inp_local;
      pthread_mutex_unlock( & mutex);
    }
  }
  return EXIT_SUCCESS;
}

/*Main function */

int main(int argc, char * argv[]) {
  // Declare variables
  dispatch_t * dpp;
  resmgr_io_funcs_t io_funcs;
  resmgr_connect_funcs_t connect_funcs;
  iofunc_attr_t ioattr;
  dispatch_context_t * ctp;
  int id;
  pthread_attr_t attr;

  // Variable to store keyboard input
  char * keyboard_main_ip;

  // Variable to store string and integer inputs for error correction
  char str[] = "";
  int temp;

  attach = name_attach(NULL, ATTACH_POINT, 0);

  if (attach == NULL) {
    perror("failed to create the channel.");
    exit(EXIT_FAILURE);
  }

  // Check for correct number of inputs
  if (argc != 2) {
    perror("Not the correct number of arguments. Please input an integral number of beats per minute");
    exit(EXIT_FAILURE);
  }

  // While Loop to get the correct inputs from the user
  while (sscanf(argv[1], "%d", & temp) != 1) {

    if (sscanf(argv[1], "%s", str) == 1) {
      printf("The input is a string\n");
      printf("Please enter an integer number of beats per minute\n");
      fgets(argv[1], 100, stdin);
    } else {
      printf("Input not recognized\n");
      printf("Please enter an integer number of beats per minute\n");
      fgets(argv[1], 100, stdin);
    }
  }

  // Initialise the global inputs structure with the user inputs
  inputs.beatPerMin = atoi(argv[1]);
  inputs.keyboard_inp = '\0';

  dpp = dispatch_create();
  iofunc_func_init(_RESMGR_CONNECT_NFUNCS, & connect_funcs, _RESMGR_IO_NFUNCS, & io_funcs);

  iofunc_attr_init( & ioattr, S_IFCHR | 0666, NULL, NULL);
  id = resmgr_attach(dpp, NULL, "/dev/local/metronome", _FTYPE_ANY, NULL, & connect_funcs, & io_funcs, & ioattr);

  ctp = dispatch_context_alloc(dpp);

  // Clear the screen
  putchar(12);

  printf("Welcome to MA4830 Metronome Program\n");
  printf("The program is going to print a pendulum moving across the screen\n depending on the number of beats per minute\n");
  printf("Use the up arrow key to 1.2x the beats per minute \nand down arrow key to 1.2/ it and press CTRL+C to end the program\n");
  printf("The screen will clear in 10 seconds, ENJOY THE PROGRAM!\n");

  delay(10000);

  pthread_attr_init( & attr);

  // Create and start the threads
  pthread_create(NULL, & attr, & MetronomeFunc, NULL);
  pthread_create(NULL, NULL, & KeyboardFunc, NULL);

  while (1) {

    ctp = dispatch_block(ctp);
    dispatch_handler(ctp);
  }

  pthread_attr_destroy( & attr);
  name_detach(attach, 0);

  return EXIT_SUCCESS;
}

```