universidade de aveiro
theoria poiesis praxis

Departamento de Eletrónica, Telecomunicações e Informática

Curso: [8240] Mestrado Integrado em Engenharia de Computadores e Telemática

Disciplina: [41475] Sistemas Tempo-Real

Ano letivo: 2016/2017

RELATÓRIO

SIMULAÇÃO DO JOGO DO PONG EM XENOMAI

Autores:

[67851] Adriano Oliveira

[68779] Rui Oliveira

Grupo:

3

Docente:

Professor Paulo Pedreiras

Data:

terça-feira, 17 de janeiro de 2017

Conteúdos

1. Enquadramento

Pretende-se através deste relatório expor sob forma escrita, o nosso desempenho e objetivos alcançados no projeto final da unidade curricular de Sistemas Tempo-Real.

Neste relatório pretendemos explicar as técnicas por nós utilizadas para implementar o presente trabalho utilizando para tal um sistemas operativo tempo real - o xenomai.

2. Objetivos/Problema

O presente trabalho consistiu em desenvolver uma simulação utilizando para isso um sistema operativo em tempo real abordado nas aulas práticas da unidade curricular de Sistemas Tempo Real - o Xenomai. O Xenomai baseia-se numa distribuições linux e suporta modos de kernel/tempo-real com elevado desempenho. Este RTOS pode ser instalado em placas de desenvolvimento como o raspberry pi entre outros. Para além disso, o xenomai possui uma extensa e completa documentação para criação e manipulação de tarefas, assim como para enviar e receber dados através de *sockets*. Possui também suporte para desenvolvimento Hard RTOS para aplicações na camada do utilizador.

Após algumas discussões na escolha da simulação que iremos desenvolver chegamos à conclusão que poderia ser uma boa opção o popular jogo do pong. Numa primeira fase pensámos que seria demasiado simples de implementação mas depois concordam que o mais importante seria desenvolver as nossas capacidades na modulação/implementação dum jogo em tempo real, podendo contudo tentar adicionar novas funcionalidade ao jogo.

O jogo do pong foi o primeiro videojogo mais lucrativo da história, dando origem a este novo setor da indústria. Foi desenvolvido por Nolan Bushnell e Ted

Dabney na forma de uma consola ligado a um monitor, movido a moedas, tal como mostra a figura seguinte.

Figura 1: Primeira versão do jogo do pong

O pong é um jogo eletrônico normalmente de duas dimensões que simula um jogo de ténis de mesa. O jogo controla duas barras verticais movendo-as verticalmente, uma no lado esquerdo e outra do lado direito correspondente a cada jogador. Um dos lados pode ser controlado através de um algoritmo inteligente que consiga, na medida do possível, resolver o principal objectivo do jogo: não deixa que a bola ultrapasse a barra. A bola aumenta de velocidade cada vez que é rebatida, reiniciando a velocidade caso algum dos jogadores não acerte na bola. O objetivo de cada jogador é fazer mais pontos que seu adversário, fazendo com que o adversário não consiga retornar a bola para o seu lado.

A nossa simulação do jogo do pong terá as seguintes caracteristicas:

- Permitirá disputar o jogo entre dois utilizadores
- Permitirá disputar o jogo entre um utilizador e o computador
- Pontuação máxima que permite declarar qual o utilizador vence o jogo é 10

Neste projeto foram utilizadas as seguintes tecnologias:

- Xenomai 2.6.4 + kernel linux 3.14.17
- Linguagem C
- SDL 1.6

Todo o código fonte encontra-se disponível no seguinte repositório criado para o efeito: https://github.com/ruipoliveira/realtime-pong-xenomai

3. Aspetos específicos

Nesta secção iremos mostrar alguns aspectos específicos de implementação, tais como as estruturas utilizadas, principais métodos criados e métodos de interação com o utilizador.

Temos duas estruturas básicas que identificam cada componente principal do jogo: a bola e a barra, sendo que no caso da barra existem duas instâncias, para o lado direito e outra para o lado esquerdo.

Estrutura de dados: bola

```
typedef struct ball_s {
      int x, y; /* position on the screen */
      int w,h;   /* ball width and height */
      int dx, dy; /* movement vector */
} ball_t;
```

A bola é identificada pela sua posição na janela(x e y), dimensões (w e h) e vetor de movimento (dx e dy). A função **`draw_ball()`** permite desenhar a bola, que neste caso é um quadrado (`SDL_FillRect`) na janela, através da sua posição e respectivas dimensões.

Estrutura de dados: barras

```
typedef struct paddle {
      int x,y;   /* bar position in the window*/
      int w,h;   /* Dimensions of the bars*/
} paddle_t;


static paddle_t paddle[2] /* paddle rigth and left*/
```

Cada barra é identificada pela posição na janela (x e y) e pela dimensão de cada barra (w e h). São criadas duas instâncias da barra, uma para ser localizada à esquerda da janela e outra à direita. A função **`draw_paddle()`** permite desenhar as duas barras e adicioná-las à janela principal.

O método `init_ball()` permite inicial as posições iniciais e tamanhos dos elementos dos jogo (bola e barras). A bola é um quadrado com dimensão 10 e as barras dimensões com dimensão 10x50.

Métodos necessários para a interface em SDL

De seguida iremos apresentar os métodos necessários para a criação da interface gráfica utilizando SDL.

- **`draw_background():`** permite desenhar o fundo da interface gráfica. É usado o construtor `SDL_FillRect` disponibilizado pelo SDL.
- **`draw_net():`** permite desenhar o tracejado ao centro que divide as duas áreas de jogo.

- **`draw_game_over(int p):`** permite apresentar na interface quem ganhou o jogo, sendo passado como argumento para este método o id do jogador

- **`draw_player_0_score():`** permite colocar na janela a pontuação do jogador do lado esquerdo.
- **`draw_player_1_score():`** permite colocar na janela a pontuação do jogador do lado direito.

Métodos mais relevantes

De seguida iremos descrever os principais métodos mais relevantes no nosso software, sem que estes sejam tarefa.

- **`check_score():`** verifica que se o número máximo de pontuação possível foi atingido
-
- **`check_collision(ball_t a, paddle_t b):`** verificar se existiu alguma colisão entre a bola e uma barra. Este método é utilizado na tarefa da cinemática que iremos abordar mais à frente.

Técnicas de controlo

As interações podem ser feitas da seguinte forma:

- Utilizador 1
  - Encontra-se sempre ativo
  - Corresponde à barra do lado direito
  - Pressionar tecla keyUP do teclado para subir a barra
    - `keystate[SDLK_UP]`
  - Pressionar tecla keyDown do teclado para descer a barra
    - `keystate[SDLK_DOWN]`
- Utilizador 2
  - Só está ativo quando existem dois jogadores a competir, caso o jogo seja contra o computador este não funciona
  - Corresponde à barra do lado esquerdo
  - Pressionar tecla W do teclado para subir a barra
    - `keystate[SDLK_w]`
  - Pressionar tecla S do teclado para descer a barra
    - `keystate[SDLK_s]`
- Para terminar o programa o utilizador terá que pressionar a tecla Q do teclado

Para subir ou descer a barra é necessário verificar se esta já não pode subir ou descer mais. No caso da subida, ao pressionar uma tecla é incrementado 10 unidades à posição y da barra. Na descida, é decrementada 10 unidades à posição y da barra.

4. Abordagem

Nesta secção serão apresentadas as principais soluções relacionadas com a implementação no xenomai.

Processamento inicial para Xenomai

Para a utilização do xenomai foram criadas duas funções para esse efeito.

- **`init_xenomai():`** usa `mlockall(MCL_CURRENT|MCL_FUTURE)` para evitar trocas de memória para este programa e executa `rt_print_auto_init(1)` caso a tarefa não o fizer.
- **`startup():`** permite declarar o período de cada tarefa existente, criar a própria tarefa através do método `rt_task_create()` e dar-lhe início através do método `rt_task_start()`

Tarefas criadas/usadas em Xenomai

Através de todas as tarefas será possível consultar os tempos máximos e mínimos.

Tarefa da cinemática
`task_cinematics_code(void *task_period_ns)`

Esta tarefa permite definir toda a cinemática associada ao movimento da bola do jogo. Nesta mesma tarefa é verificado se a bola passa a zona limite das barras que possibilitará definir se ocorreu "golo" ou não, permitindo incrementar a variável que nos retornará a pontuação final de cada jogador. No caso de existir colisão com uma das barras, esta tarefa também permitirá definir o aumento de velocidade aplicada, quanto maior o ângulo de embate maior será a velocidade de projeção.

Tarefa de movimentação da barra
`void task_move_paddle_code(void *task_period_ns)`

Esta tarefa permite definir o mecanismo de controlo da barra, para um utilizador ou para dois (no caso de não ser escolhida a opção contra o computador). Através do SDL é verificado se o utilizador pressionou uma determinada tecla, se tal acontecer é o valor da posição y da barra é alterado.

Tarefa de movimentação da barra com inteligência artificial
`void task_move_paddle_ai_code(void *task_period_ns)`

Esta tarefa apenas é activada quando o utilizador escolhe competir contra o "computador". Permite definir o mecanismo automático de controlo do movimento da barra. A posição y da barra é atualizada conforme a posição y da bola, deslocando-se paralelamente.

Períodos

Os períodos aplicados neste programa são os que se apresentam de seguinda. Decidimos que a tarefa da cinemática terá um período maior, possibilitando que esta seja executada com maior prioridade em relação às restantes.

Tarefa da cinemática
```
#define TASK_CINEMATICS_PERIOD_NS 100000000 // Task period, in ns
```

Tarefa de movimentação da barra
```
#define TASK_MOVE_PADDLE_PERIOD_NS 99000000 // Task period, in ns
```

Tarefa de movimentação da barra com inteligência artificial
```
#define TASK_MOVE_PADDLE_AI_PERIOD_NS 99000000 // Task period, in ns
```

5. Resultados

Execução e requisitos

Requisitos:

- Xenomai 2.6.4 + kernel linux 3.14.17 (ou equivalente)
- SDL interface 1.2

Compilação:

- make

Execução:

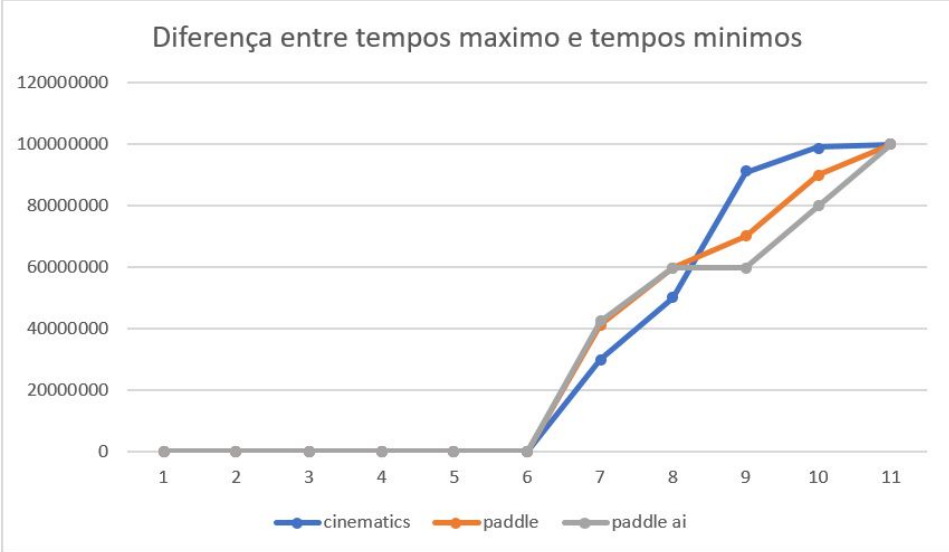
- ./pong
  - multiplayer (player 1: keyUP, keyDown; player 2: W, S)
- ./pong debug
  - Mostra posição da bola e teclas pressionadas
- ./pong computer
  - Começa jogo contra o "computador" (player 1: keyUP, keyDown)
- ./pong computer debug
  - Mostra posição da bola e teclas pressionadas no caso do jogo ser contra o "computador"

Tempos medidos

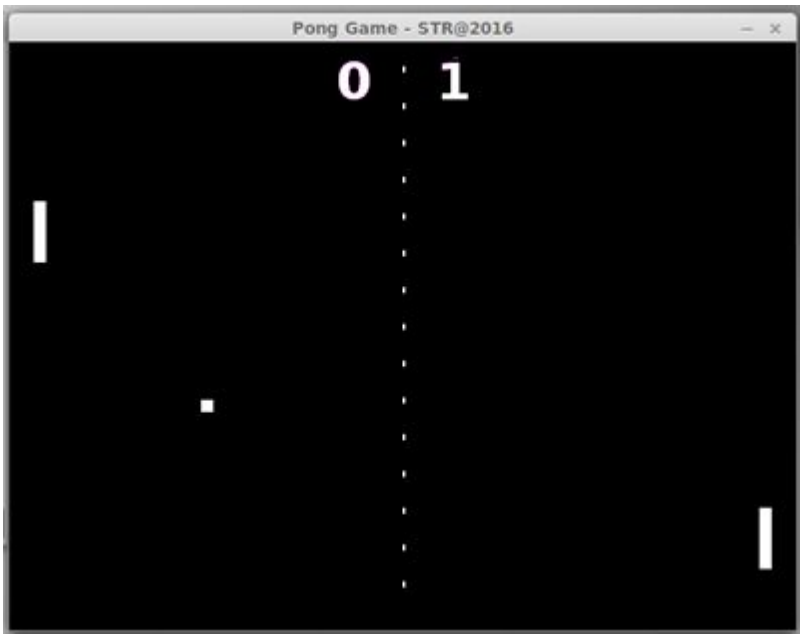
Obtivemos os resultados que de seguida se apresentam:

Tarefa	**GAP (ns)**	**MAX (ns)**	**MIN (ns)**
cinematics	100000626	0	100000626
paddle	100003413	0	100003413
paddle ai	100004074	0	100004074
cinematics	100000462	0	100000462
paddle	99999281	0	99999281
paddle ai	99998743	0	99998743
paddle	89056558	0	89056558
paddle ai	89067899	0	89067899
cinematics	119091505	0	119091505

Diferença entre tempos máximo e tempos mínimos



Interface gráfica obtida



6. Estado final e principais conclusões

Chegado ao final deste relatório, é nossa intenção efetuar uma retrospetiva da evolução do mesmo, tendo em conta os problemas com que nos deparámos, e principais conclusões retiradas. No decurso da realização deste trabalho deparámo-nos com alguns problemas que limitaram o nosso desempenho nomeadamente a instalação do xenomai que demorou mais tempo do que seria desejado e expectável. Este problema não nos permitiu focar desde início na resolução do trabalho proposto.

Apesar dos problemas encontrados, foi possível desenvolver uma simulação em tempo real em que as tarefas implementadas cooperam todas paralelamente não sendo necessário qualquer mecanismo de sincronização de tarefas pois estas estão isoladas umas das outras.

Como conclusão, pensamos que os resultados se encontram de acordo com o esperado, sendo que conseguimos um software a cooperar com um sistema operativo em tempo real.

7. Referências

- https://xenomai.org/ (Documentação Xenomai)
- http://ppedreiras.av.it.pt/resources/str1617/praticas/install-Xenomai-2.6.4-Mint-17.2-README.txt (Guia de instalação)
- http://ppedreiras.av.it.pt/resources/str1617/praticas/xenomai-class-guide.pdf
- https://en.wikipedia.org/wiki/Xenomai
- https://pt.wikipedia.org/wiki/Pong