ISSN 1870-4069

JScheduling: A Graphical Interface for Applying a Process Scheduling Algorithm

Adriana Hernández-Beristain[1], Erika Annabel Martínez-Mirón[1], Mariano Larios-Gómez[1], Javier Caldera-Miguel[2], Luis Angel Zamarripa-Almazan[1]

[1] Benemérita Universidad Autónoma de Puebla, Puebla, Mexico

[2] Universidad Politécnica de Puebla, Puebla, Mexico

adriana_beristain@hotmail.com, mlarios777@gmail.com, erika.a.mtzm@gmail.com, javiercmiguel@hotmail.com

Abstract. Among the tasks that scheduling algorithms perform are the provision of time to each node and processes management, either in a task queue or in a list. It is possible, via a command line terminal, to track the performance of these algorithms. Nevertheless, a visual environment that facilitates this tracking would be very helpful. This work describes the design and implementation of a didactic graphical interface for a distributed embedded software, which allows the visual representation of a process scheduling algorithm in a virtual mobile distributed system.

Keywords. Distributed system, mobile device, embedded software.

1 Introduction

This work describes the implementation of a graphical interface, named JScheduling, for an embedded software; this software shows the use of a process scheduling algorithm that allows to a supercomputer the allocation of its resources optimally among the different nodes connected to it.

A well-known process scheduling algorithm is Simple Linux Utility for Resource Management (SLURM) [1], which is open code and is implemented in many supercomputers based in Linux. This scheduler gives the nodes a time to execute their tasks, manages the processes launched by each node and considers a queue of pending tasks for accessing resources. Besides, SLURM optimizes the nodes' allocation with an algorithm centered in the Hilbert curve [2].

Another process scheduling algorithm, which will be named as “fan”, gives the nodes the same amount of time to execute their tasks and, after reaching a consensus, decide the access to the resources.

Sometimes, the understanding of these algorithms can represent a difficulty due to the associated abstraction. So, the implementation of JScheduling to represent a mobile distributed system, where the user can manage the nodes in a simple, easy and transparent way, as well as visualize how each node executes its processes, becomes a very useful and didactic tool.

The following sections describe the implementation of JScheduling in order to represent, configure, and communicate the nodes, as well as how the fan process scheduling algorithm is used in order to give access to the resources to each node.

2 Development of JScheduling

During the requirement analysis and specification phase, the functionalities identified were: a) the node network creation (being the nodes the mobile devices), and b) the obtaining of the required information to implement a process scheduler under a virtual environment. The following subsections describe how these functionalities were achieved.

2.1 Node Representation

In order to represent a virtual mobile device some considerations were taken into account: 1) a simple design and, 2) to insert as many mobile devices as possible. The first was achieved by using a simple rectangle to represent a node, straight lines for its connections (see Figure 1) and the use of the right click of the mouse for editing options (more details in Section 2.2). It is important to mention that on the back-end of the software, a single adjacency list with four elements (transmitter node, receiver node, bridge node and router node), was used to store the connections. The latter was accomplished combining the computer's date and time and generating a unique identifier for each node.

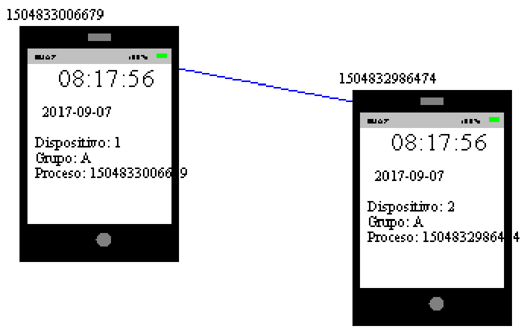


Fig. 1. Representation of two nodes with a single connection.

In addition, for representing the mobile devices and their respective resource requests, Petri networks [3] were used because of their graph design. A graph is defined as a triplet G = <P, R, A>, where P is the set of processes used by the scheduling algorithm, R is the set of resources used in a distributed environment and A is the set of edges that relate one to another process [4]. Figure 2 depicts an example where processes p1 and p2 request the resources, but it is the process p1 that gets the critical section.

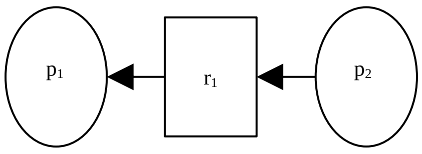


Fig. 2. Example of Petri nets notation.

2.2 Node Configuration

For the creation and manipulation of each node on the workspace, a contextual control menu can be used (right click) to display a set of possible actions: a) New SmartPhone; b) Dragging; c) Connection; d) Individual elimination; e) Overall elimination; f) Intercommunication - JNI y g) Cancellation. Figure 3 shows the menu highlighting the Java Native Interface (JNI) function, that allows the native calls for the communication between the embedded software and the underneath operative system (OS).

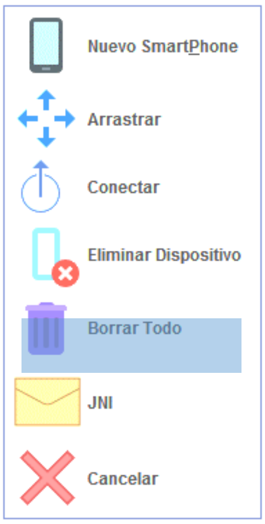


Fig. 3. Contextual control menu.

2.3 Communication between Nodes

The core functionality of JScheduling is the communication between the virtual mobile devices through an information flow between two layers, the graphic layer and the low level layer. For this reason, a multiplatform with JNI and the supercomputer's OS (Minix) was implemented.

Minix is a modular operative system in which services are considered as processes, in comparison to other operative systems, where services are treated as simple calls to the system [5]. In Minix, any kind of processes can communicate between them through primitives to exchange information by means of messages.

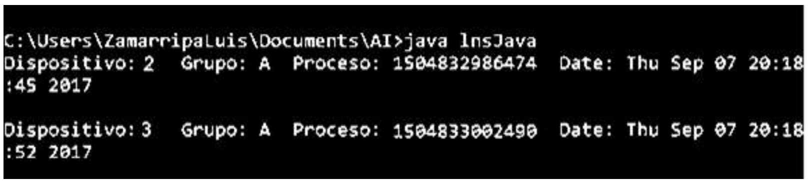
In JScheduling, the information exchanged between the nodes includes the node's identifier, the execution time (start, final) and the quantum assigned by the system to the process. The code lines shown in Figure 4 correspond to the function that requests for time in the process planning algorithm and assign the respective ID to the node.

```
  #include <jni.h>
  #include <stdio.h>
  #include <string.h>
  #include <time.h>
  JNIEXPORT void JNICALL Java_DelayTime
    (JNIEnv * env, jobject jobj,jstring i){
   int ID;
   time_t rawtime;
    struct tm * timeinfo;
    time ( &rawtime );
    timeinfo = localtime ( &rawtime );
    const  char  *str  =  (*env)->GetStringUTFChars(env,  i,
0);
    printf("Dispositivo: %d  Grupo: A  Proceso: %s  Date:
%s  \n",
    ID++, str,asctime (timeinfo));
  }
```

Fig. 4. ID generation for each node in the low level layer.

3 Functioning of JScheduling

For representing the process scheduling, a driver capable of using real resources was implemented as a DLL file. When the JNI option is selected in the contextual menu, the driver requests the ID processes of the connected nodes from the supercomputer working on real time. Then a command line terminal is used to verify that the exchange of messages between the connected nodes was correct (see Figure 5).



```
C:\Users\ZamarripaLuis\Documents\AI>java lnsJava
Dispositivo: 2   Grupo: A   Proceso: 1504832986474   Date: Thu Sep 07 20:18
:45 2017

Dispositivo: 3   Grupo: A   Proceso: 1504833002490   Date: Thu Sep 07 20:18
:52 2017
```

Fig. 5. Exchange of messages between emitter (Dispositivo 2) and receptor (Dispositivo 3) nodes.

3.1 Characteristics of the Evaluation Environment

Because a personal computer did not have enough resources for testing the creation of more than five mobile devices, there was the need to use a supercomputer, whose characteristics are described next: The Cuetlaxcoapan supercomputer of the LNS is composed of a standard calculation cluster with Intel Xeon processors and a cluster with Intel Xeon Phi Knights Landing processors with 228 Thin calculation nodes (5472 total cores). Each node contains 2 Intel Xeon E5-2680 v3 processors (Haswell) at 2.5 GHz, 12 cores per processor / 24 total cores, 128 GB of DDR4 memory at 2133 MHz, 2 Gigabit Ethernet network interfaces and an InfiniBand FDR 56 Gbps network interface. Storage of 1.2 PB of total space for disk storage. Finally, there is a Gigabit Ethernet network for managing the hardware of the supercomputer and the provisioning of software to the nodes.

4 Results

The implementation of an interface that allows to represent any mobile distributed system was achieved. The mobile devices are introduced as single rectangles, the straight lines denote the connection between them, and the information inside each node correspond to the resources requested to the system. The interface is centered in graphs, edges and relations between the mobile devices.

Besides the graphical representation of the mobile distributed network, it is possible to visualize the functioning of a process planning algorithm (fan) by means of the information displayed in each mobile device representation. Figure 6 shows how the graphical interface and the process planning algorithm are working together.

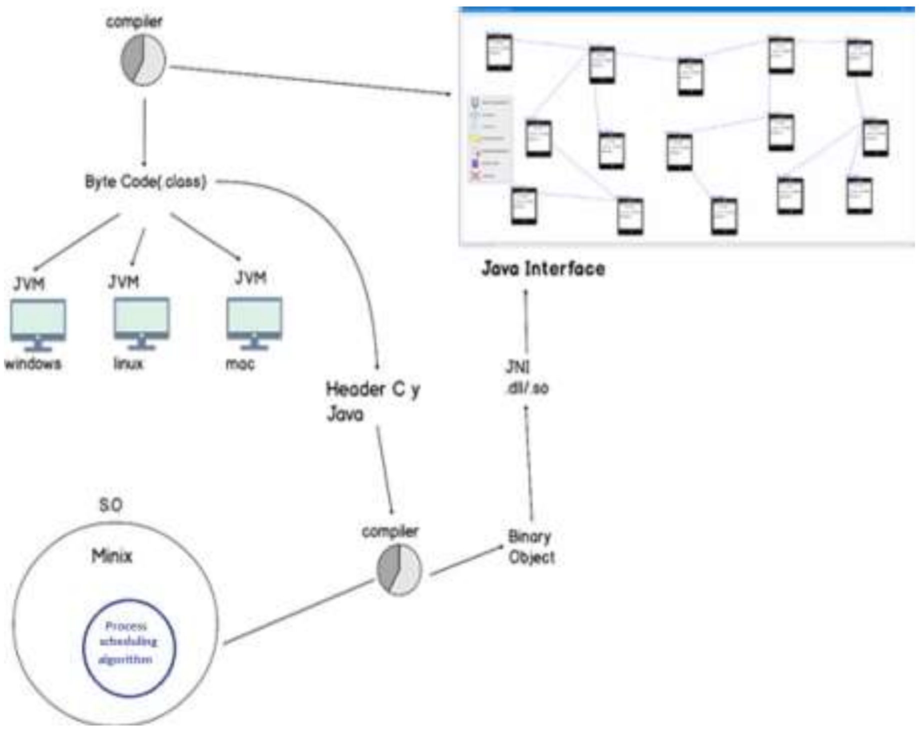


Fig. 6. The graphical interface and the process scheduling algorithm.

5 Conclusions and Future Work

The implemented graphical interface allows the visual representation of a mobile distributed system in a simple, easy and transparent way. At the same time, it is possible to visualize how a fan process scheduling algorithm is executed through the exchange of information between the nodes by means of labels inserted in each node.

So far, just one process can be executed by each node, but it is planned to modify the functions to allow the generation of more processes. Also, the use of other process scheduling algorithms is considered.

Acknowledgments. The authors thankfully acknowledge the computer resources, technical expertise and support provided by the Laboratorio Nacional de Supercómputo del Sureste de México, CONACYT national laboratories network.

References

1. Trinitis, C., Weidendorfer, J.: Co-Scheduling of HPC Applications. 71–73 (2017)

2. Costa, L., Oliveir. P.: An elistist genetic algorithm for multiobjective optimization. 309–310 (2003)
3. Tanenbaum, A. S., Wetherall, D.J.: Computer networks. 232–237 (214)
4. Lee, J. S.: A Petri net design of command filters for semiautonomous mobile sensor networks. IEEE Transactions on Industrial Electronics, 55(4), 1835–1841 (2008)
5. Herder, J. N., Bos, H., Gras, B., Homburg, P., Tanenbaum, A. S.: MINIX 3: A highly reliable, self-repairing operating system. ACM SIGOPS Operating Systems Review, 40(3), 80–89 (2006)