CSE-511 OPERATING SYSTEMS DESIGN

Project 1: Synchronization with Path Expressions

Mustafa Goktan Gudukbay - mfg5472
Adarsh Singh - ars7407

1 Introduction

We followed the paper authored by Campbell and Habermann when we were implementing our project [1]. To implement the additional Child-Care Problem and the Monkey-Crossing Problem, we used multiple path expressions and extra behaviors.

2 Design and Implementation

Path expressions enforce order and synchronization behavior on operations. Each operation has a prologue and an epilogue part. Before an operation can start, it must perform the prologue. After the operation is finished, it must execute the epilogue part. According to Campbell, path expressions have four rules. Path expressions can contain sequences, selections, simultaneous executions, and repetitions. Sequences enforce order between two actions, and it is specified with a semicolon. Selections permit only one of two actions to occur and are specified with a plus sign. Simultaneous execution allows the operations specified to execute concurrently and is specified by braces. Repetition allows a path expression to repeat after it is completed once, and it is specified by the path and end keywords. A path expression can contain multiple repetitions, and each repetition is controlled by a separate controller mechanism.

We first separated each *path*... *end* expression to parse the expression. Then we followed the algorithm in Section 4 of [1] for each *path*... *end* expression. Algorithm 1 summarizes the procedure for parsing each path expression.

Algorithm 1 Parsing The Path Expression

Require: path $<$ expression $>$ end

1. **Stage:** Select a unique semaphore $S1$, initialized to one, and replace path $<$ expression $>$ end by $P(S1) <$ expression $> V(S1)$. Continue to Stage 2 to parse the remaining $<$ expression $>$.

2. **Stage:** Do one of the following (a sequence or a selection must be parsed before simultaneous execution)

   (a) **Sequence:** The $<$ expression $>$ is in the format:
   Prologue $<$ expression_1 $>;<$ expression_2 $>$ Epilogue.
   Select a unique semaphore $S2$, initialized to zero, and replace the $<$ *expression* $>$ with these two expressions:

   i. Prologue $<$ expression_1 $> V(S2)$
   ii. $P(S2) <$ expression_2 $>$ Epilogue

   Continue to parse the two expressions separately by returning back to Stage 2.

   (b) **Selection:** The $<$ expression $>$ is in the format:
   Prologue $<$ expression_1 $> + <$ expression_2 $>$ Epilogue.
   Replace the $<$ *expression* $>$ with these two expressions:

   i. Prologue $<$ expression_1 $>$ Epilogue
   ii. Prologue $<$ expression_2 $>$ Epilogue

   Continue to parse the two expressions separately by returning back to Stage 2.

   (c) **Simultaneous Execution:** The $<$ expression $>$ is in the format:

   Prologue {expression} Epilogue.

   The Prologue must be a $P(S_i)$ and the Epilogue must be a $V(S_j)$ operation. Select a unique counter C initialized to zero, and semaphore $S3$ initialized to one. Replace the braces and the P, V operations with the following expression:

   $PP(C, S3, S_i) <$ *expression* $> VV(C, S3, S_j)$
   Continue to parse the *expression* by returning back to Stage 2.

   (d) **Procedure:** The remaining expression is a procedure name. It has a prologue and an epilogue. Prologue is going to be executed in the *enter operation* function and the Epilogue is going to be executed in the *exit operation* function in our code.

The PP operation is defined in Algorithm 2:

Algorithm 2 PP

Require: $C, S3, S_i$
  semWait(S3);
  C = C + 1;
  if $C == 1$ **then**
    semWait(S_i);
  end if
  semPost($S3$);

The VV operation is defined in Algorithm 3:

Algorithm 3 VV

Require: $C, S3, S_j$
  semWait($S3$);
  C = C - 1;
  if $C == 0$ **then**
    semPost(S_i);
  end if
  semPost($S3$);

The algorithm is recursive. Therefore, the parsing procedure can be visualized as a tree. Each leaf in the tree would correspond to an operation name and have a prologue and an epilogue. After parsing the expression, we store it in a tree in our implementation. In the enter and exit operations, we use binary search to find the corresponding leaf node and then apply the necessary prologue and epilogue procedures.

3 Child-Care Problem

We used two path expressions for this problem. Our expressions are:

1. *path*{*CaregiverArrive*; *CaregiverLeave*} *end*
2. *path*{*ChildArrive*; *ChildLeave*} *end*

The prologue and epilogue of the four operations after the parsing procedure are listed below:

1. *PP(C1, S2, S1)*$CaregiverArriveV(S3)$
2. $P(S3)CaregiverLeaveVV(C1, S2, S1)$
3. *PP(C2, S5, S4)*$ChildArriveV(S6)$
4. $P(S6)ChildLeaveVV(C2, S5, S4)$

These expressions enable multiple caregivers or children to arrive and leave concurrently. Additionally, we use an additional semaphore for the number of caregivers, which is initialized to zero at the beginning of the program. On top of the standard operations enforced by the path expression, we do extra operations using this semaphore. When a caregiver arrives, we signal on *S_noOfCaregivers* in the epilogue. The epilogue of the *CaregiverArrive* operation after modifications is given in Algorithm 4:

Algorithm 4 CaregiverArrive Epilogue

Require: $S3, S\_noOfCaregivers$
  semWait($S3$);
  semPost(*S_noOfCaregivers*);

When a caregiver leaves, it waits on *S_noOfCaregivers* in the prologue. The prologue of the *CaregiverLeave* operation after modifications is given in Algorithm 5:

Algorithm 5 CaregiverLeave Prologue

Require: $S3, S\_noOfCaregivers$
  semWait($S3$);
  semWait(*S_noOfCaregivers*);

When the first child arrives, it waits on *S_noOfCaregivers* in the prologue. The prologue of the *ChildArrive* operation after modifications is provided in Algorithm 6:

Algorithm 6 ChildArrive Prologue

Require: $C2, S5, S4, S\_noOfCaregivers$
  semWait(S5);
  C2 = C2 + 1;
  if $C == 1$ **then**
    semWait($S4$);
    semWait(*S_noOfCaregivers*);
  end if
  semPost($S5$);

When the last child leaves, it signals on *S_noOfCaregivers* in the epilogue. The epilogue of the *ChildLeave* operation after modifications is given in Algorithm 7:

Algorithm 7 ChildLeave Epilogue

Require: $C2, S5, S4, S\_noOfCaregivers$
  semWait(S5);
  C2 = C2 - 1;
  if $C == 0$ **then**
    semPost($S4$);
    semPost($S\_noOfCaregivers$);
  end if
  semPost($S5$);

When a child arrives or leaves, we use the counter that is already defined in the PP and VV operations to understand if it is the first child or the last child.

4 Monkey Problem

For this problem, we defined a new rule:

$$/\mathrm{X} < expression > \setminus \ where\ X > 1$$

Our rule is the limited version of the simultaneous execution rule defined by Campbell and Habermann [1]. We permit a maximum of X processes in that expression to execute concurrently. This rule was required for the problem because we needed a way to enforce a limit on procedures that can be executed concurrently. We used a mutex and a counter initialized to 1 to achieve the desired behavior. The mutex protected the access on the counter variable. The counter keeps track of the number of processes doing this operation. Suppose a process wants to execute that operation when the counter's value equals X. In that case, we use a condition variable to wait until we get a finished signal from other processes. Besides, we used an additional semaphore S_i. The first process to execute the procedure inside the braces must get that semaphore in the prologue. When the last process executes that procedure, it signals the semaphore in the epilogue. The algorithms for *Prologue* and *Epilogue* operations are given in Algorithms 8 and 9, respectively.

Algorithm 8 Prologue

Require: $Counter, Mutex, S_i, X, Condition$
  lock(mutex);
  while $C == X$ **do**
    condWait(Condition);
  end while
  Counter += 1;
  if $C == 1$ **then**
    semWait(S_i);
  end if
  unlock(mutex);

Algorithm 9 Epilogue

Require: $Counter, Mutex, S_i, X, Condition$
  lock(mutex);
  Counter -= 1;
  if $C == 0$ **then**
    semPost(S_i);
  end if
  unlock(mutex);

The path expression for the monkey problem is:

$$path\ /5\ EastCrossing \setminus +\ /5\ WestCrossing \setminus end$$

The selection enables either the east or west sides to cross at a time. We used our rule to limit the number of monkeys on the rope.

5 Workload Distribution

M. Goktan Gudukbay: Worked on parser design, *INIT_SYNCHRONIZER*, *ENTER_OPERATION*, and *EXIT_OPERATION*, additional problems and report.

Adarsh Singh: Worked on additional problems, *ENTER_OPERATION*, *EXIT_OPERATION* and tested the code with new input files.

References

[1] R. H. Campbell and A. N. Habermann, "The specification of process synchronization by path expressions," in *Conference on Operating Systems, OS 1974. LNCS*, vol. 16. Springer, 1974, pp. 89–102.