



1 Earliest deadline first-algoritmen

I detta dokument belyses schemalägnings algoritmen Earliest deadline first (EDF). Redogörelsen inleds med en översikt och går vidare med ett djupare bevis på hur algoritmen fungerar.

1.1 En översikt

EDF-algoritmen ordnar prioriteten mellan olika processer genom att undersöka dess deadline. Deadline är den absolut senaste tid då processen måste exekverats färdig. Den process som har tidigast deadline får högst exekveringsprioritet, dvs får köra först.[Joh98]

Det finns en indelning till; med eller utan förkörsrätt. Det innebär att en process kan ges möjlighet att avbryta en annan process då dess deadline ligger tidigare än det nu exekverande. På engelska heter detta preemption. Non-preemptive är när denna typ av avbrott inte tillåts.

Prioriteterna är dynamiska vilket innebär att deadlines kan ändras under pågående programexekvering. Algoritmen är heuristisk, dvs den är självlärande. [C. 97]

1.2 Förutsättningar för algoritmen

EDF-algoritmen är optimal för en ensam processor (Jackson's rule), på så sätt att om inte EDF-algoritmen kan schemalägga en uppsättning processer så kan ingen algoritm göra detta.

Algoritmen kan bevisas genom att undersöka följande ekvation:

$$\sum_{i=1}^n \frac{e_i}{P_i} \leq 1 \quad (1)$$



2 Matematisk bevis för periodiska processer och förkörsrätt.

Under denna rubrik kommer bevisen för att EDF-algoritmen alltid kan schemaläggas för en processor som har en utnyttjandegrad u som är

$$u = \sum_{i=1}^n \frac{e_i}{P_i} \leq 1 \quad (2)$$

Som hjälp används motsägelsebevis. Detta gäller alltså under föutsättning att följande *inte* är sant;

$$u > 1 \quad (3)$$

eller att

$$t < \min \left\{ P + d_{max}, \max_{1 \leq i \leq n} \{P_i - d_i\} \right\} \quad (4)$$

sådant att

$$h_T(t) > t \quad (5)$$

2.1 $u > 1$

För det första skall det bevisas att ingen optimal schemaläggning kan äga rum om inte (2) gäller, dvs att vissa processer kommer att missa sin deadline om

$$\sum_{i=1}^n \frac{e_i}{P_i} > 1 \quad (6)$$

För att bevisa detta utnyttjas det faktum att processornas utnyttjandegrad inte kan överstiga ett (100%).

För att genomföra beviset införs begreppet *lcm*, least common multiple. *lcm* är det lägsta talet vilket är jämt delbart med alla tal i en talserie.

$$P = lcm\{P_1, P_2, \dots, P_n\}$$

Det betyder att P ger den första tidpunkten då alla processer samtidigt har sina periodtidsslut.



ℓ_i är antalet perioder, P_i , som en process hinner exekvera inom den totala perioden P .

$$\ell_i = \frac{P}{P_i}$$

Den totala exekveringstiden över ett intervall P kommer att ges av

$$\sum_{i=1}^n \ell_i e_i = \sum_{i=1}^n \frac{P}{P_i} e_i = P \sum_{i=1}^n \frac{e_i}{P_i}$$

Enligt (6) är

$$\sum_{i=1}^n \frac{e_i}{P_i} > 1$$

vilket betyder att

$$P \underbrace{\sum_{i=1}^n \frac{e_i}{P_i}}_{>1} > P$$

Detta ger att vissa processer kommer att missa sin deadline då den totala exekveringstiden för processerna överskrider processorns totala exekveringstid, P .
V.S.B.

I detta fallet studeras alla processers periodtid, P_i som en talserie.
Antag att det finns en tid t_{UL} . Summera alla hela processer som exekveras innan t_{UL} och multiplicera med varje process exekveringstid.

$$\sum_{i=1}^n \left\lfloor \frac{t_{UL}}{P_i} \right\rfloor \cdot e_i > t_{UL} \quad (7)$$

där $\lfloor n \rfloor$ ger heltalet av n . Dividera med t_{UL} på båda sidor och

$$\sum_{i=1}^n \frac{e_i}{P_i} > 1 \quad (8)$$

V.S.B.

Anta fortsättningsvis att (2) gäller.
Även om (2) så finns det en uppsättning processer för vilken det inte går att finna en optimal schemaläggning eftersom en minst process kommer att missa sin deadline.

Låt t_{UL} vara den tid då en deadline för första gången kommer att missas. Om en process missar sin deadline innebär det att processorn varit upptagen under hela intervallet från start till t_{UL} , $[0, t_{UL}]$.



Det finns två fall

- *Ingen* av processerna har sin deadline *efter* tiden t_{UL} .
- Vissa av processerna *har* deadline efter tiden t_{UL} .

Den första punkten innebär att en process deadline inträffar innan eller samtidigt som t_{UL} .

Innan tiden t_{UL} hinner en process exekveras

$$\left\lfloor \frac{t_{UL}}{P_i} \right\rfloor$$

gånger. $\lfloor n \rfloor$ trunkerar resultatet av n vilket är det antal gånger en period helt hinner exekvera innan tiden t_{UL} . Exekverings tiden för processen i ges då av

$$\left\lfloor \frac{t_{UL}}{P_i} \right\rfloor e_i$$

Missas deadline innebär det att exekveringstiden för alla processer kommer att var större än den tillgängliga, alltså t_{UL} som innebär att

$$\sum_{i=1}^n \left\lfloor \frac{t_{UL}}{P_i} \right\rfloor e_i > t_{UL}$$

Trunkeringen av $\frac{t_{UL}}{P_i}$ kan ses som att talet subtraheras med ett decimaltal lika stort som decimaldelen av kvoten. Detta betyder att $\frac{t_{UL}}{P_i} > \left\lfloor \frac{t_{UL}}{P_i} \right\rfloor$ och därav följer

$$\sum_{i=1}^n \left(\frac{t_{UL}}{P_i} \right) e_i > t_{UL}$$

vilket, efter division med t_{UL} på båda sidor, ger

$$\sum_{i=1}^n \left(\frac{e_i}{P_i} \right) > 1$$



2.2 t_{UL}

$h_T(t_{UL})$ är summan av all exekveringstid för en uppsättning processer som har sin deadline innan tiden t_{UL} . Beviset går ut på att visa att en uppsättning inte går att schemalägga om

$$h_T(t_{UL}) < \min \left\{ P + d_{max}, \max_{1 \leq i \leq n} \{P_i - d_i\} \right\}$$

sådant att

$$t_{UL} < h_T(t_{UL})$$

Som hjälp används att

$$h_T(t_{UL} + \alpha) > t_{UL} + \alpha \Rightarrow h_T(t_{UL}) > t_{UL}$$

för alla

$$t_{UL} \geq d_{max}$$

Hjälpsatsen bevisas på följande sätt:

Högra delen adderas med α i båda sina led

$$h_T(t_{UL}) + \alpha > t_{UL} + \alpha$$

$$h_T(t_{UL}) + \alpha = \sum_{i=1}^n e_i \left(\left\lfloor \frac{t_{UL} - d_i}{P_i} \right\rfloor + 1 \right) + \alpha \quad (9)$$

Antalet perioder som ryms i tiden

$$\left\lfloor \frac{t_{UL} - d_i}{P_i} \right\rfloor$$

ger oss antalet perioder *efter* d_1 varför en period måste adderas. Vidare gäller att $u \leq 1$ varför

$$\alpha \geq \alpha \sum_{i=1}^n \left(\frac{e_i}{P_i} \right)$$

Insatt i (9)

$$h_T(t_{UL}) + \alpha \geq \sum_{i=1}^n e_i \left(\left\lfloor \frac{t_{UL} - d_i}{P_i} + 1 \right\rfloor \right) + \sum_{i=1}^n \left(\frac{e_i}{P_i} \right) \alpha$$

$$h_T(t_{UL}) + \alpha \geq \sum_{i=1}^n e_i \left(\left\lfloor \frac{t_{UL} - d_i + \alpha}{P_i} + 1 \right\rfloor \right)$$



Alltså gäller

$$\begin{aligned}h_T(t_{UL}) + \alpha &\geq h_T(t_{UL}) + \alpha > t_{UL} + \alpha \\h_T(t_{UL}) &> t_{UL}\end{aligned}$$

V.S.B

Direkt ur detta fås att om $u \leq 1$ och att $h_T(t_{UL}) < P + d_{max}$ så kommer inte uppsättningen av processer kunna schemalägga eftersom $h_T(t_{UL}) > t$.

Vidare antas att en uppsättning processer inte kan schemaläggas trots att $u \leq 1$. Dessutom gäller att $h_T(t_{UL}) > t_{UL}$ vilket ger

$$t_{UL} < d_{max}$$

eller

$$t_{UL} < \max_{1 \leq i \leq n} \{P_i - d_i\} \frac{u}{1 - u}$$

För att bevisa detta sätts t_{UL} att vara $t > d_{max}$

Enligt tidigare resonemang är

$$\begin{aligned}h_T(t_{UL}) &\leq \sum_{i=1}^n e_i \frac{t_{UL} - d_i}{P_i} + 1 \\h_T(t_{UL}) &\leq \sum_{i=1}^n e_i \frac{t_{UL} - d_i + P_i}{P_i} \\h_T(t_{UL}) &\leq t_{UL} \sum_{i=1}^n \frac{e_i}{P_i} + \sum_{i=1}^n e_i \frac{P_i - d_i}{P_i}\end{aligned}$$

För att bryta ut $\frac{e_i}{P_i}$ ur båda termerna så används alltid den längsta tiden av $P_i - d_i$ alltså $\max\{P_i - d_i\}$, vilket innebär att svaret alltid kommer att vara större än eller lika med högerledet.

$$h_T(t_{UL}) \leq \sum_{i=1}^n \frac{e_i}{P_i} \left(t_{UL} + \max_{1 \leq i \leq n} \{P_i - d_i\} \right)$$



Eftersom $h_T(t_{UL}) > t_{UL}$ så gäller

$$t_{UL} < \sum_{i=1}^n \frac{e_i}{P_i} \left(t_{UL} + \max_{1 \leq i \leq n} \{P_i - d_i\} \right)$$

$$t_{UL} < \sum_{i=1}^n u \left(t_{UL} + \max_{1 \leq i \leq n} \{P_i - d_i\} \right)$$

$$t_{UL} < \sum_{i=1}^n u \left(t_{UL} + \max_{1 \leq i \leq n} \{P_i - d_i\} \right)$$

$$t_{UL} < ut_{UL} + u \max_{1 \leq i \leq n} \{P_i - d_i\}$$

$$t_{UL} - u \cdot t_{UL} < u \max_{1 \leq i \leq n} \{P_i - d_i\}$$

$$t_{UL}(1 - u) < u \max_{1 \leq i \leq n} \{P_i - d_i\}$$

$$t_{UL} < \frac{u}{1 - u} \max_{1 \leq i \leq n} \{P_i - d_i\}$$

Fortfarande gäller

$$h_T(t_{UL}) > t_{UL}$$

3 Företrädes och uteslutningsvillkor

Vi har innan förutsatt att task är oberoende och får avbryta varandra. Vi lämnar nu dessa förutsättningar och tittar på några andra heuristiska funktioner (algoritmer).

3.1 PREC1

Med PREC1 algoritmen schemaläggs tasken i förväg efter exekveringsordning. Här väntar man inte på deadline innan nästa task får börja exekveras. På så vis klarar fler task sin deadline. Dessutom förkortas tiden för att behandla samtliga tasks.

Antag att ett antal tasks har följande exekveringstider och deadlines:

T_i	e_i	D_i	T_i	e_i	D_i
1	3	6	2	3	7
3	2	20	4	5	21
5	6	27	6	6	28

Figure 1: Exempel på olika tasks exekveringstiders samt deadlines.

Dessutom har tasken följande prioritetsordning:

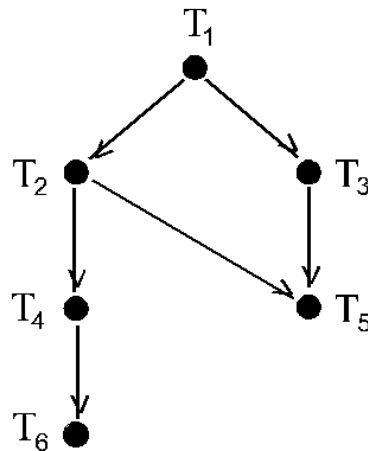


Figure 2: Krav på exekveringsordning.

dvs T_1 måste ha exekverats innan T_2 och T_3 .
 T_2 måste ha exekverats innan T_4 , T_5 och T_6 .
 T_3 måste ha exekverat innan T_5 .

Till att börja med läggs det task med högst deadline ut, dvs T_6 som har deadline vid $t = 28$. Därefter följer det task med näst högsta deadline (T_5), vilket placeras efter det att T_6 skall ha hunnit exekveras. I detta fallet placeras T_5 :s deadline vid $t = 22$ eftersom $e_6 = 5$. Det innebär att T_5 garanterat klarar sin deadline vid $t = 27$.

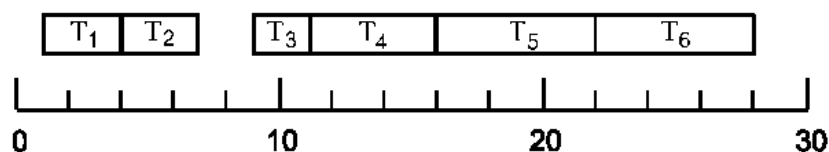


Figure 3: Task med högst deadline placeras först.

När sedan alla tasks är utplacerade komprimeras alla lediga tider.

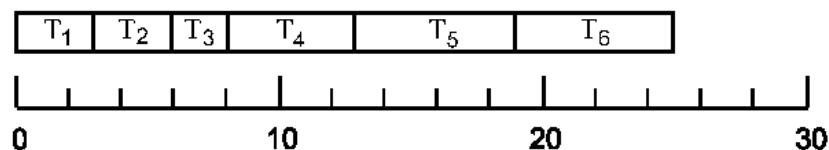


Figure 4: Färdigt schema enligt PREC1 algoritmen.

I detta fallet innebär det att T_1 läggs direkt vid $t = 0$, och har exekverats färdigt vid $t = 3$. Då kan T_2 exekveras direkt efter och vara färdig vid $t = 6$ och på så vis klara sin deadline ($D_2 = 7$). Därefter kan T_3 börja exekvera vid $t = 6$ osv...

3.2 AND/OR villkor

En ordningsgraf för tasks kan reduceras genom att tasken kan tilldelas olika villkor.

- AND task - kan inte börja exekvera innan tidigare task exekverat färdigt.
- OR task - kan börja exekvera när något av tasken exekverat färdigt.

Efter att ordningsgrafen är modifierad, färdigställs schemalagningen med PREC1 eller någon annan algoritm.

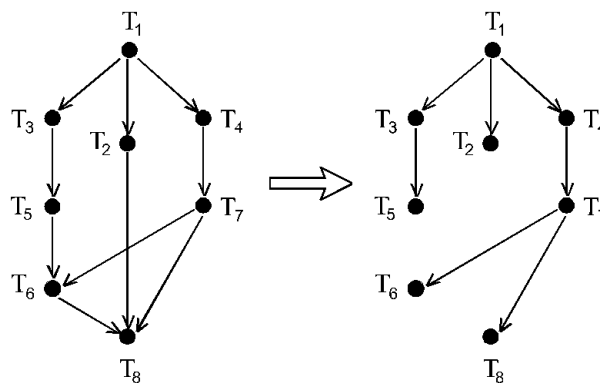


Figure 5: Förenkling av ordningsgraf.

I figuren ovan motsvarar den vänstra grafen originalgrafen, där T_6 , T_7 och T_8 av typen OR-task. Detta innebär att grafen kan förenklas till motsvarande högra grafen. Denna förenklings algoritmen kallas för MINPATH (minimum path). Fördelen med MINPATH är att den transformerar AND/OR problem till vanliga standard problem.

För mer komplicerade strukturer måste release time (r_i) för tasken modifieras för att klara av alla deadlines. Klarar vi alla deadlines så är schemat färdigt. Annars får vi gå igenom alla tänkbara kombinationer innan vi hittar ett lämpligt schema.



Schemat kallas giltigt om:

- Processorn inte är idel, om det finns ett eller flera tasks som är redo att exekveras.
- Schemat tillåter constraints som preemption, precedence och exclusion.

Utan constraints används EDF som vanligt.

Om två task har samma deadline väljs det task som har längst exekveringstid!

Positiv lateness för ett task innebär att det har missat sin deadline.

Max lateness för schemat motsvarar det task med störst lateness.

Resultatet blir optimal schemaläggning med en algoritm som är komplicerad och svår att implementera.

References

- [C. 97] C. M. Krishna, Kang G. Shin. *Real-Time Systems*. McGraw-Hill International editions, 1997.
- [Joh98] John A. Stankovic, Marco Spuri, Krithi Ramamritham, Giorgio C. Buttazzo. *Deadline Scheduling For Real-Time Systems*. Kluwer Academic Publishers, 1998.